Getting Started with Recursion

Logistics: PollEV

Lecture Participation

- Starting next Monday, we will be using the website PollEV to ask questions in lecture.
- If you provide thoughtful answers to those questions, you'll get participation credit for the day.
  - "Thoughtful" doesn't mean "correct." It's okay to have a wrong answer!
- If you can't attend lectures, or would prefer not to have participation count toward your grade, you can opt out and shift the weight to your final exam in Week 4.

Lecture Participation

- We’ll use today to dry-run PollEV questions.
- Let’s start with the following warm-up question:

> Make a book recommendation!
>
> **Answer at https://cs106b.stanford.edu/pollev**

- A few of my own recommendations:
  - Nonfiction: “Uncommon Carriers” by John McPhee.
  - Short stories: “Interpreter of Maladies” by Jhumpa Lahiri.
  - Fiction: “American Pastoral” by Philip Roth.

Outline for Today

- ***Recursive Functions***
  - A new problem-solving perspective.
- ***Recursion on Strings***
  - Featuring cute animals!

Thinking Recursively

Factorials!

- The number ***n factorial***, denoted ***n!***, is defined as

$$n \times (n - 1) \times \ldots \times 3 \times 2 \times 1$$

- Here's some examples!
  - $3! = 3 \times 2 \times 1 = 6$.
  - $4! = 4 \times 3 \times 2 \times 1 = 24$.
  - $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.
  - $0! = 1$. (by definition!)
- Factorials show up in unexpected places! We'll see one later this quarter when we talk about sorting algorithms!
- Let's implement a function to compute factorials!

Computing Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Computing Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Computing Factorials

$$5! \;=\; 5 \times \underbrace{4 \times 3 \times 2 \times 1}_{4!}$$

Computing Factorials

$$5! = 5 \times 4!$$

Computing Factorials

$$5! = 5 \times 4!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$

Computing Factorials

$$5! \;=\; 5 \times 4!$$

$$4! \;=\; 4 \times 3 \times 2 \times 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times \underbrace{3 \times 2 \times 1}_{3!}$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2 \times 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2 \times 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times \underbrace{2 \times 1}_{2!}$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1 \times 0!$$

Computing Factorials

$$5! = 5 \times 4!$$
$$4! = 4 \times 3!$$
$$3! = 3 \times 2!$$
$$2! = 2 \times 1!$$
$$1! = 1 \times 0!$$
$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1 \times \mathbf{1}$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = \mathbf{1}$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times \mathbf{1}$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = \mathbf{2}$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times \mathbf{2}$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$
$$4! = 4 \times 3!$$
$$3! = \mathbf{6}$$
$$2! = 2$$
$$1! = 1$$
$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 4 \times \mathbf{6}$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$
$$4! = \mathbf{24}$$
$$3! = 6$$
$$2! = 2$$
$$1! = 1$$
$$0! = 1$$

Computing Factorials

$$5! = 5 \times 4!$$

$$4! = 24$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times \mathbf{24}$$

$$4! = 24$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

5!	=	**120**
4!	=	24
3!	=	6
2!	=	2
1!	=	1
0!	=	1

Computing Factorials

$5! = 120$

$4! = 24$

$3! = 6$

$2! = 2$

$1! = 1$

$0! = 1$

Computing Factorials

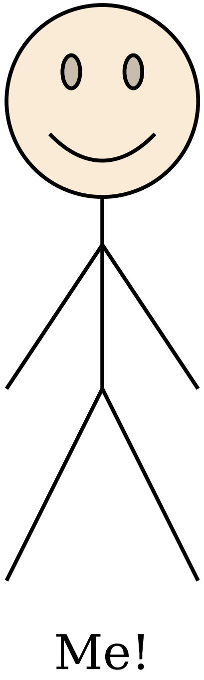
$$5! = 5 \times 4!$$
$$4! = 4 \times 3!$$
$$3! = 3 \times 2!$$
$$2! = 2 \times 1!$$
$$1! = 1 \times 0!$$
$$0! = 1$$

Another View of Factorials

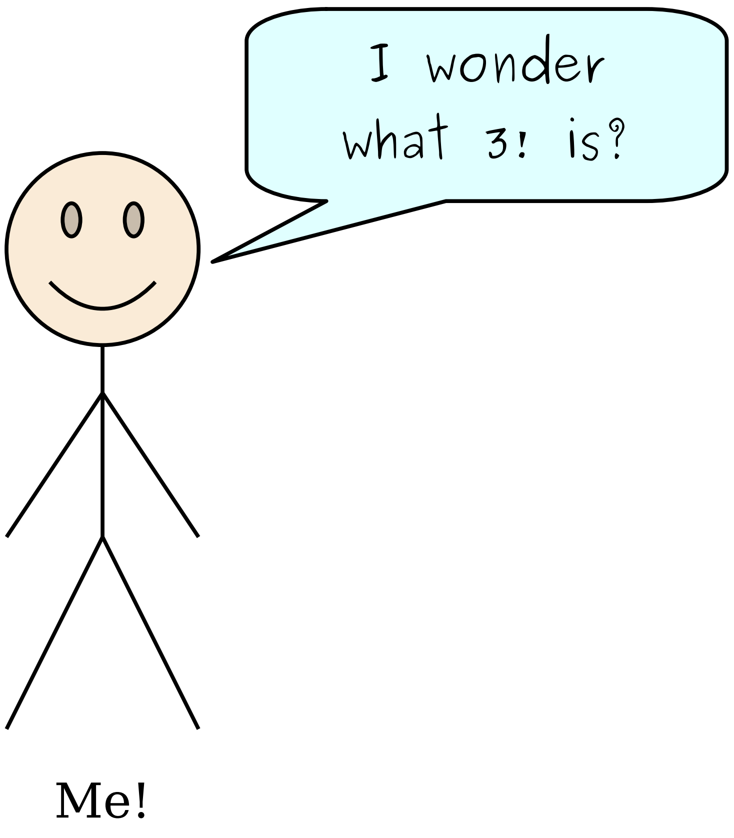
$$n! = \begin{cases} 1 & \text{if } n=0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

Alexes Compute Factorials

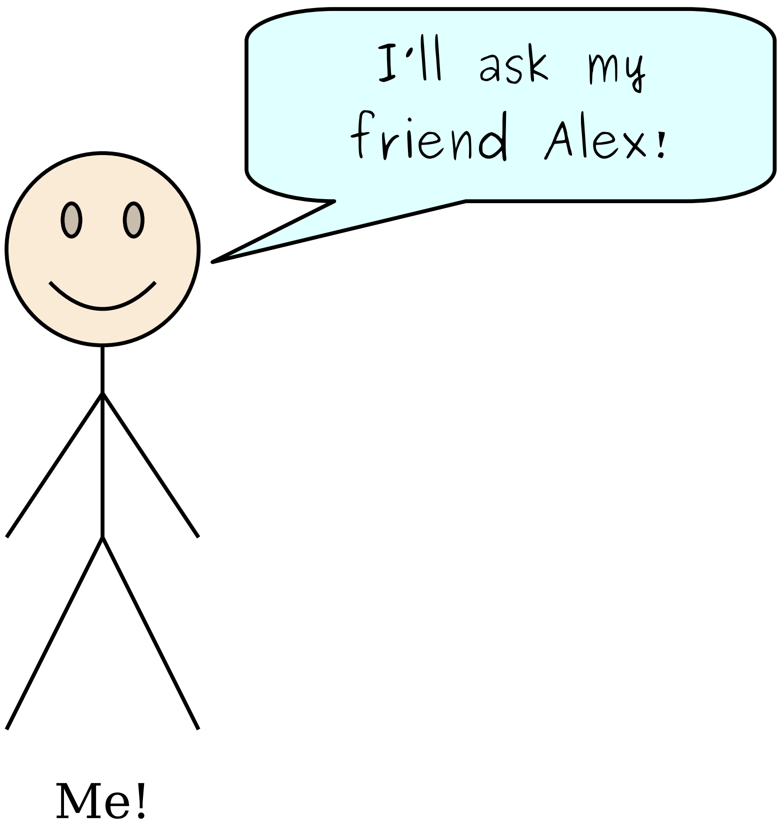
Alexes Compute Factorials



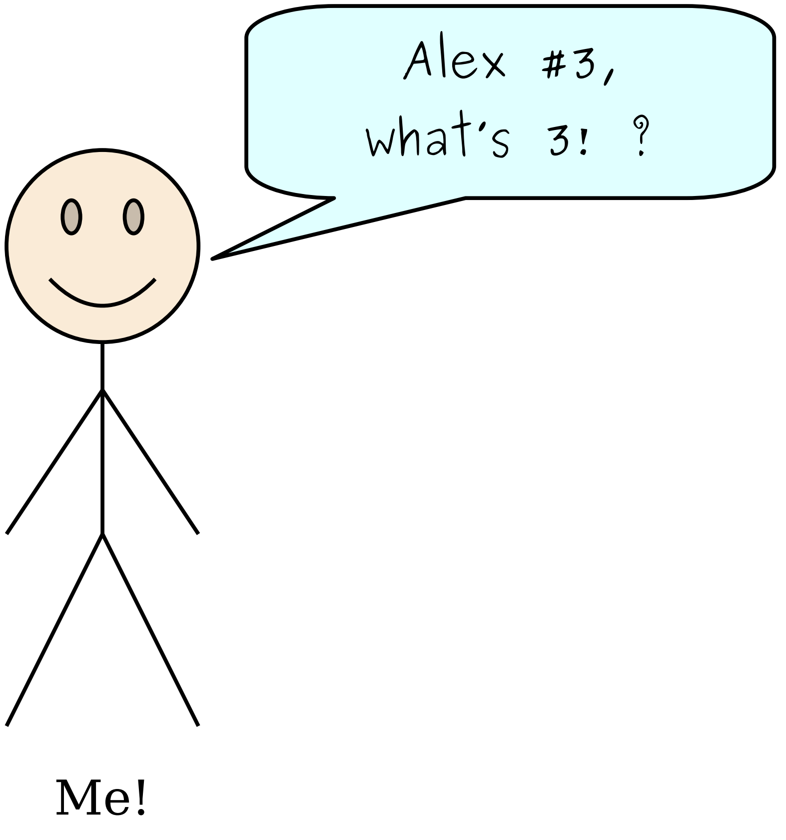
Alexes Compute Factorials



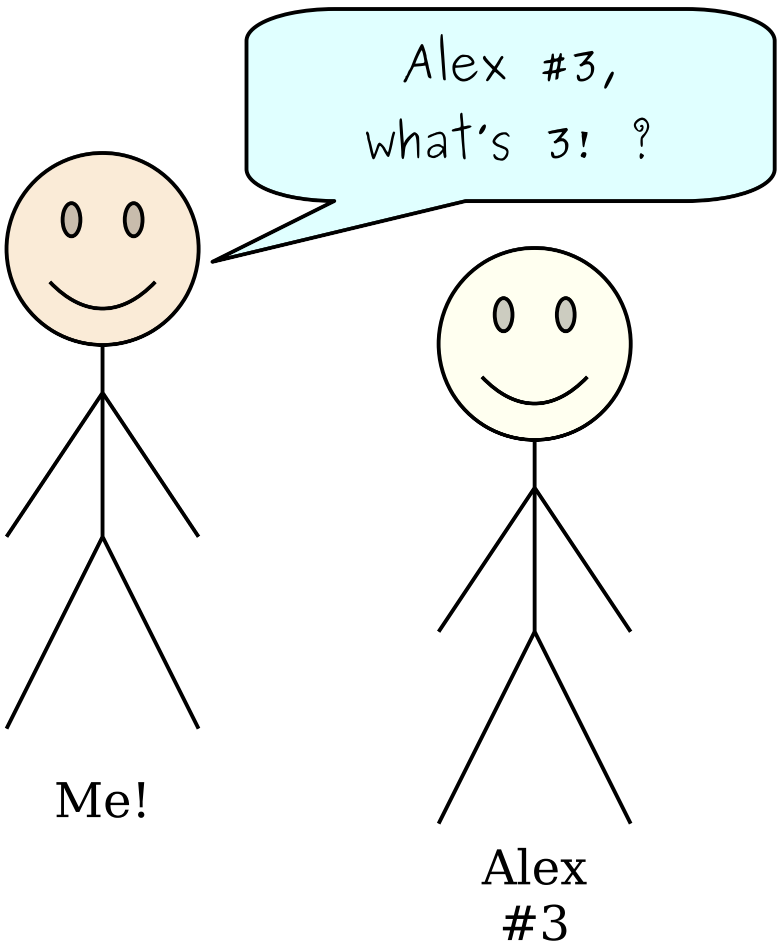
Alexes Compute Factorials



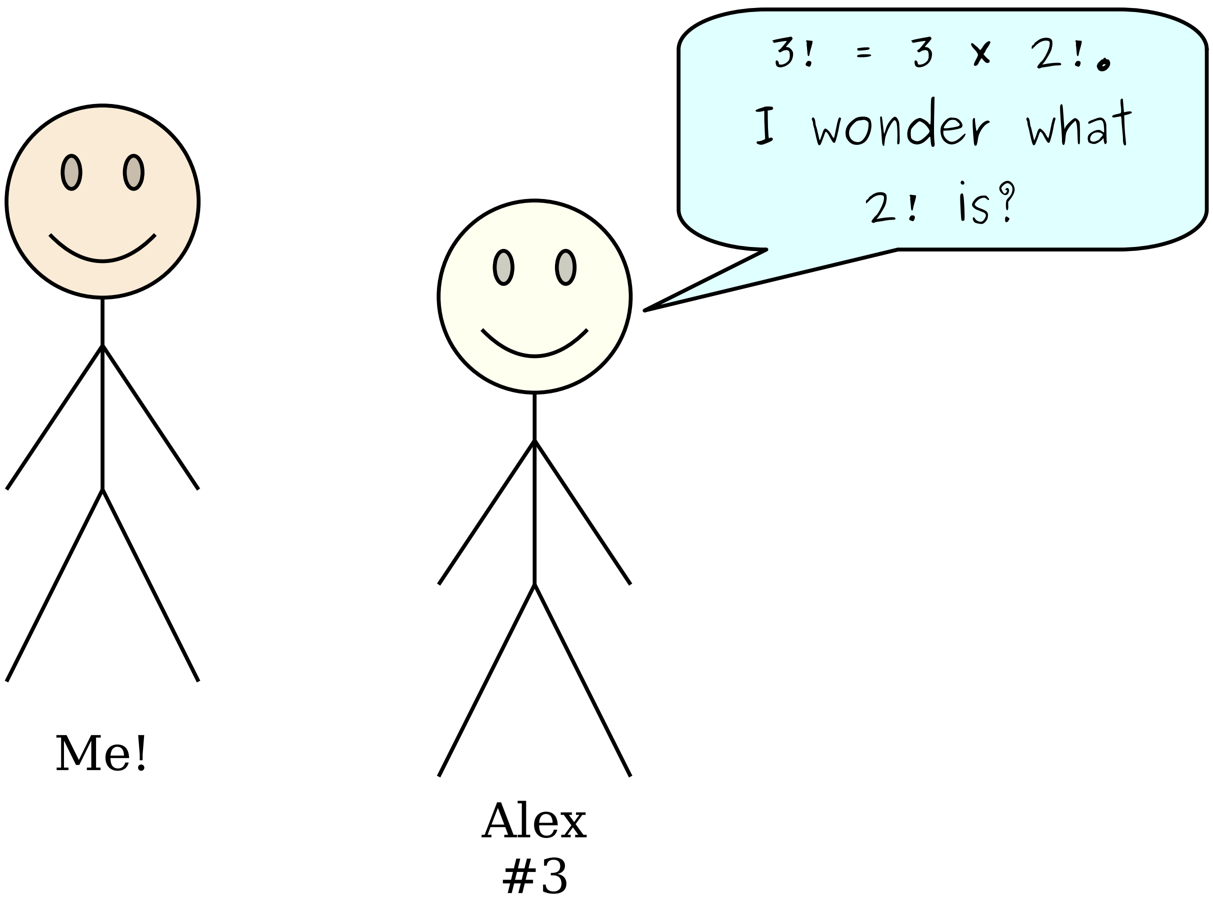
Alexes Compute Factorials



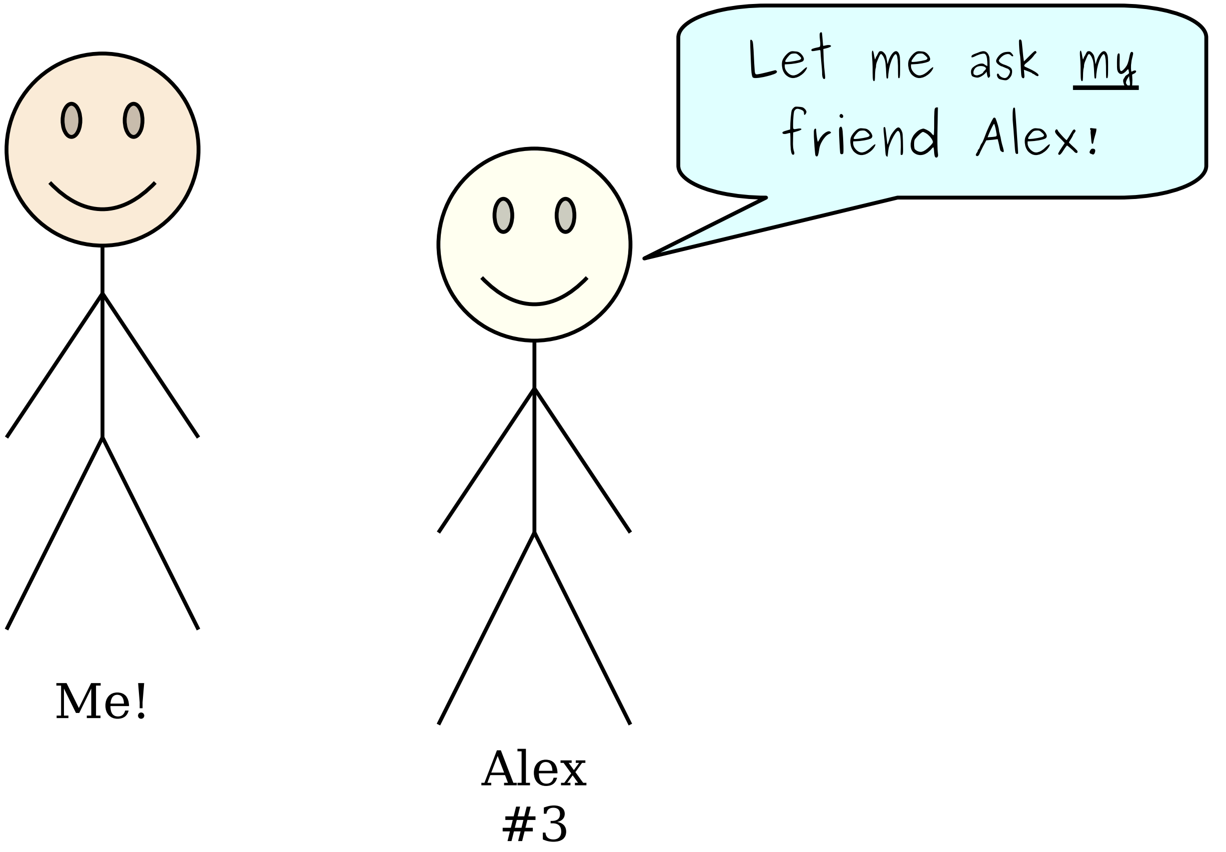
Alexes Compute Factorials



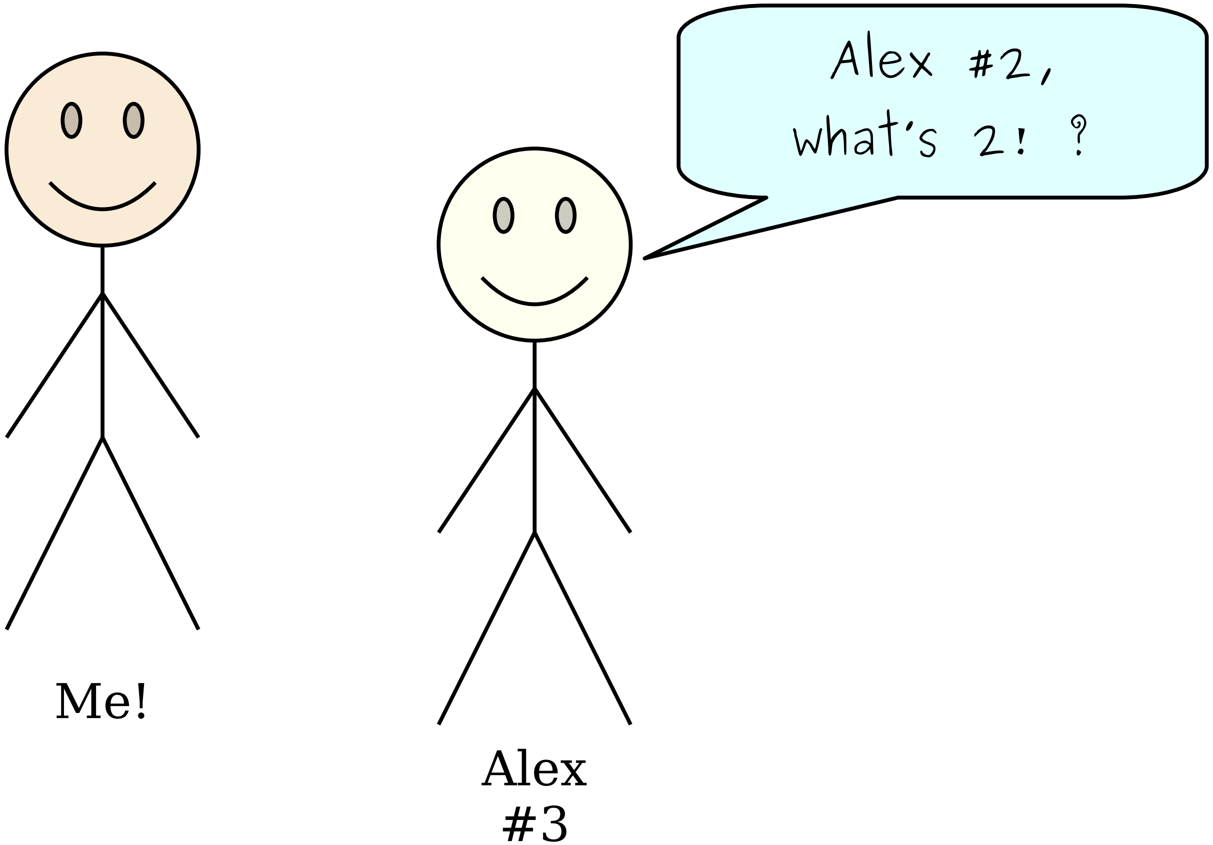
Alexes Compute Factorials



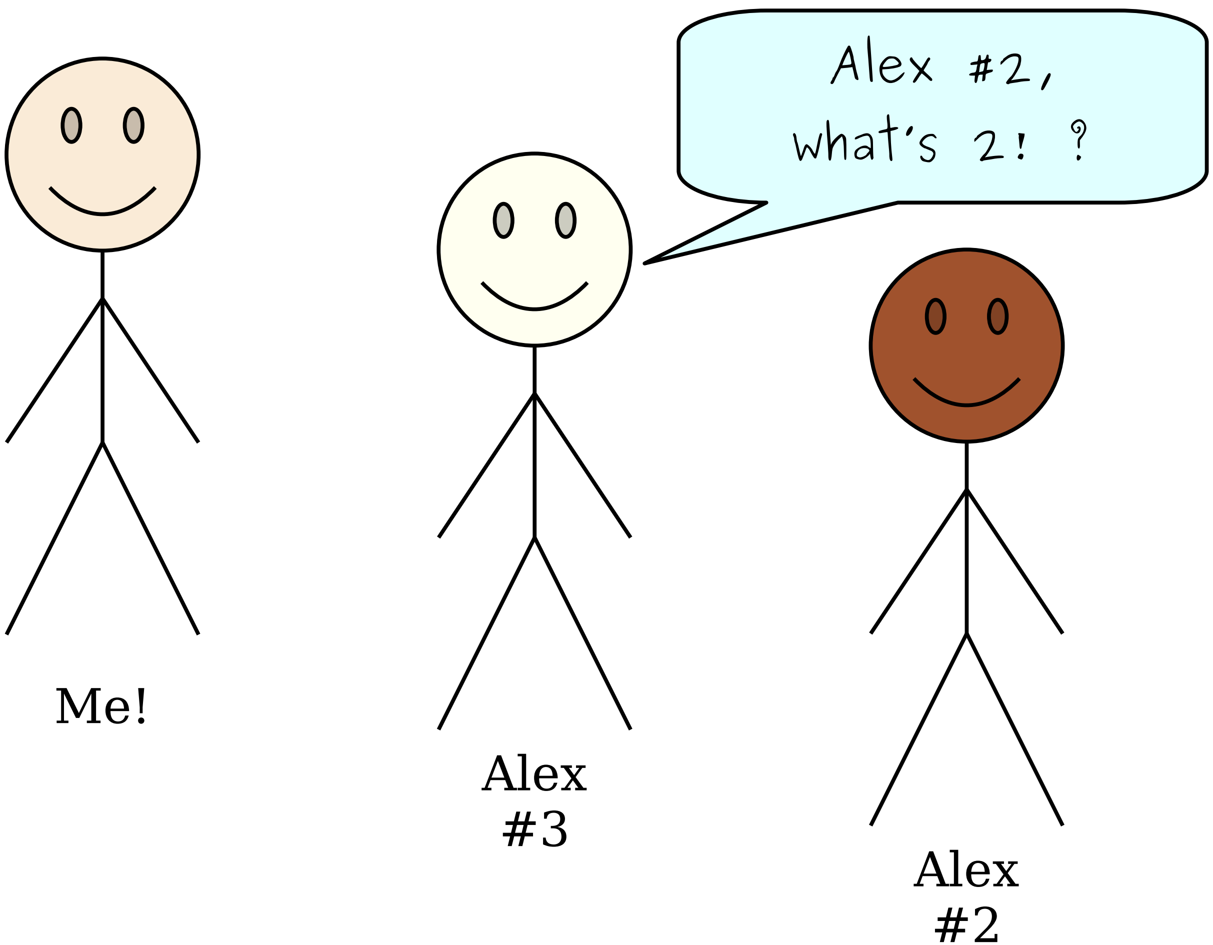
Alexes Compute Factorials



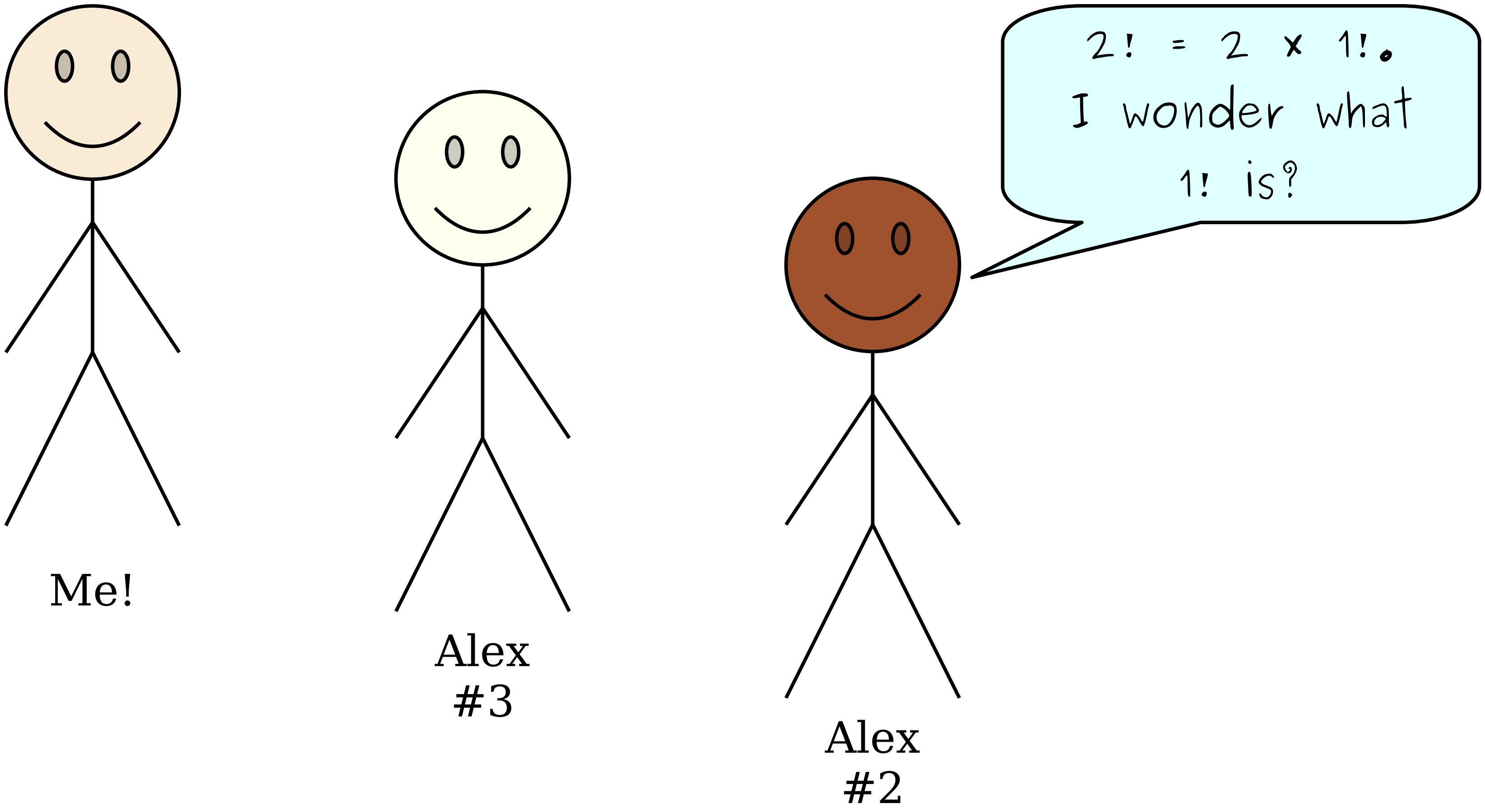
Alexes Compute Factorials



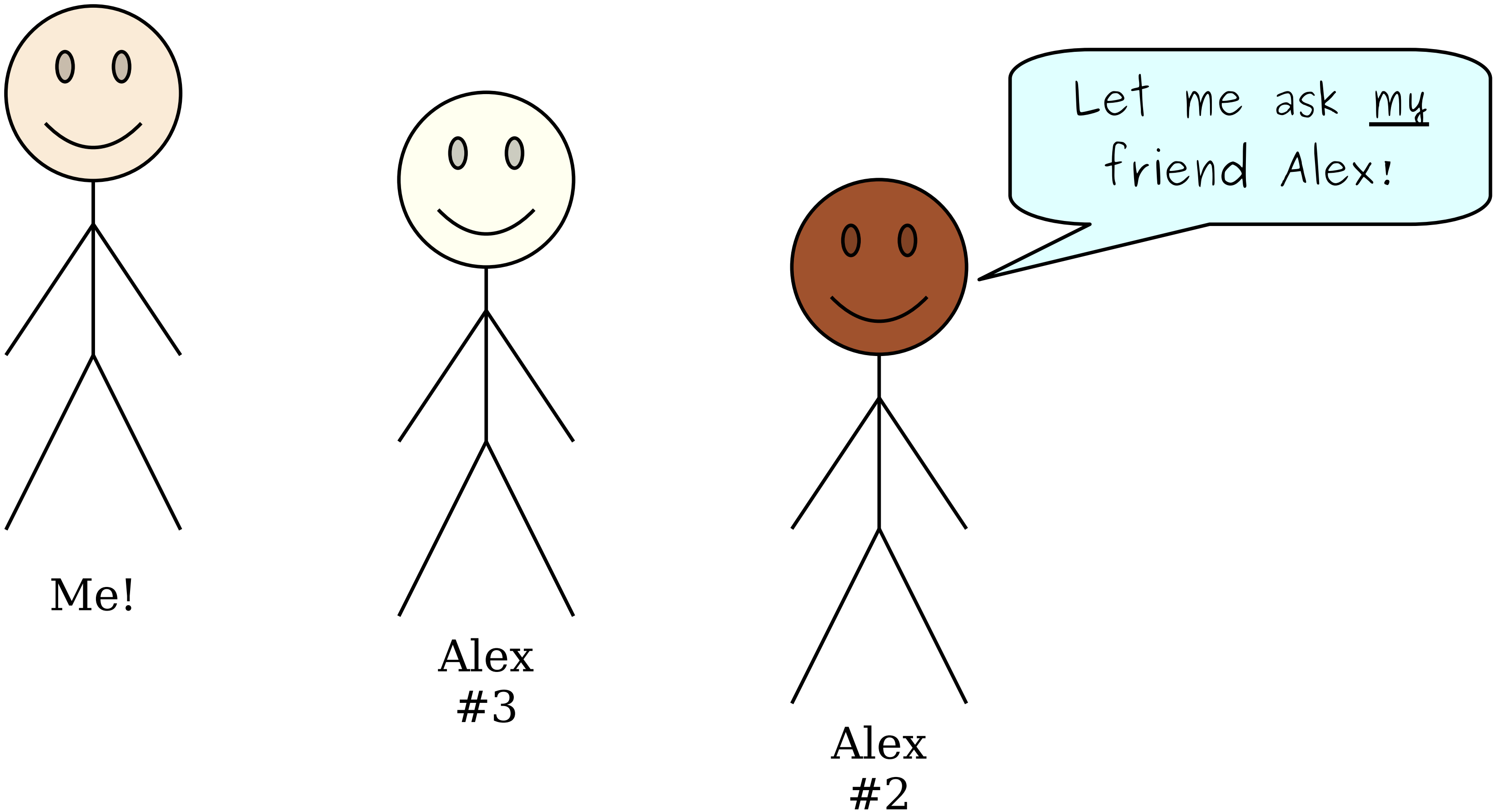
Alexes Compute Factorials



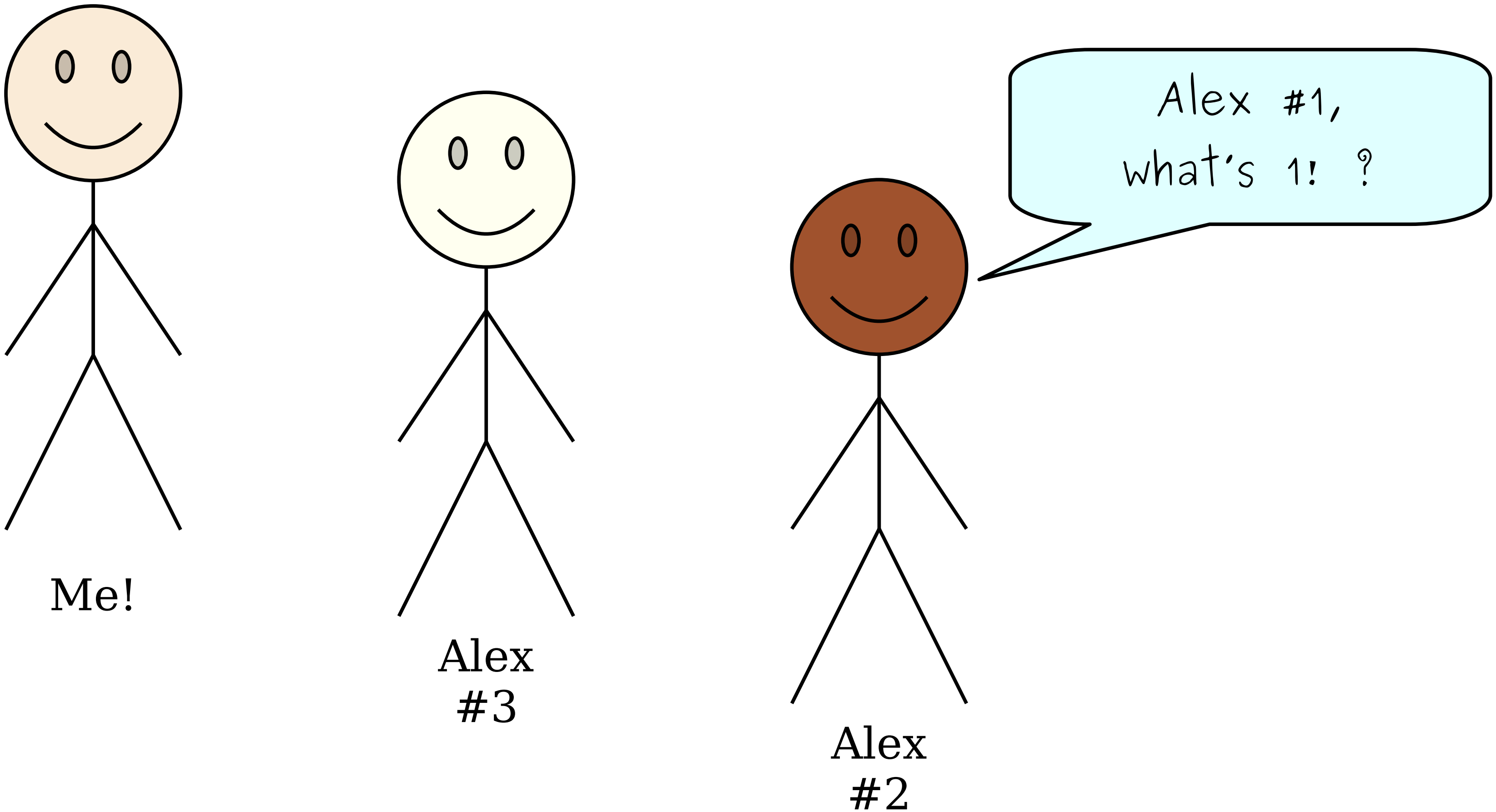
Alexes Compute Factorials



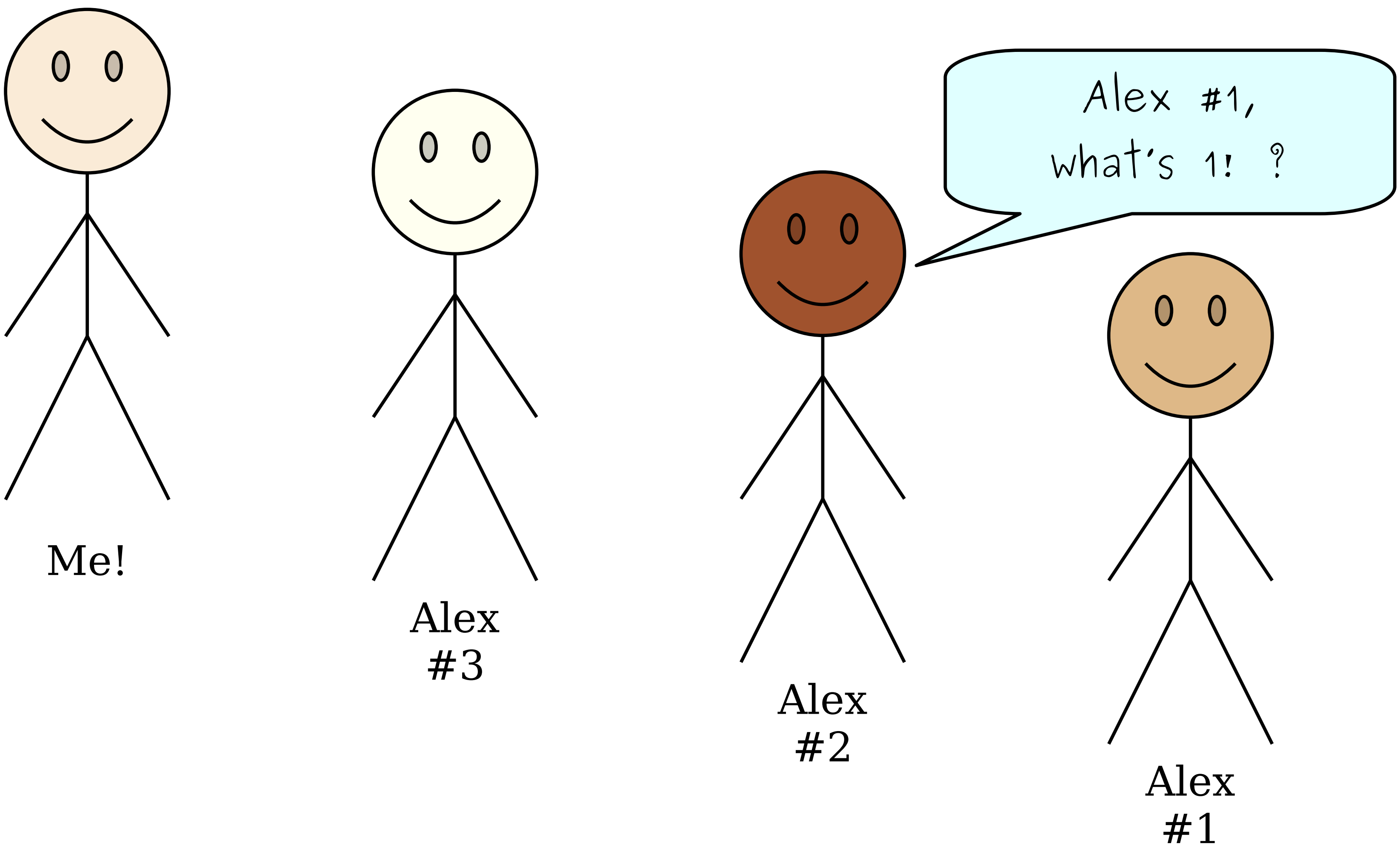
Alexes Compute Factorials



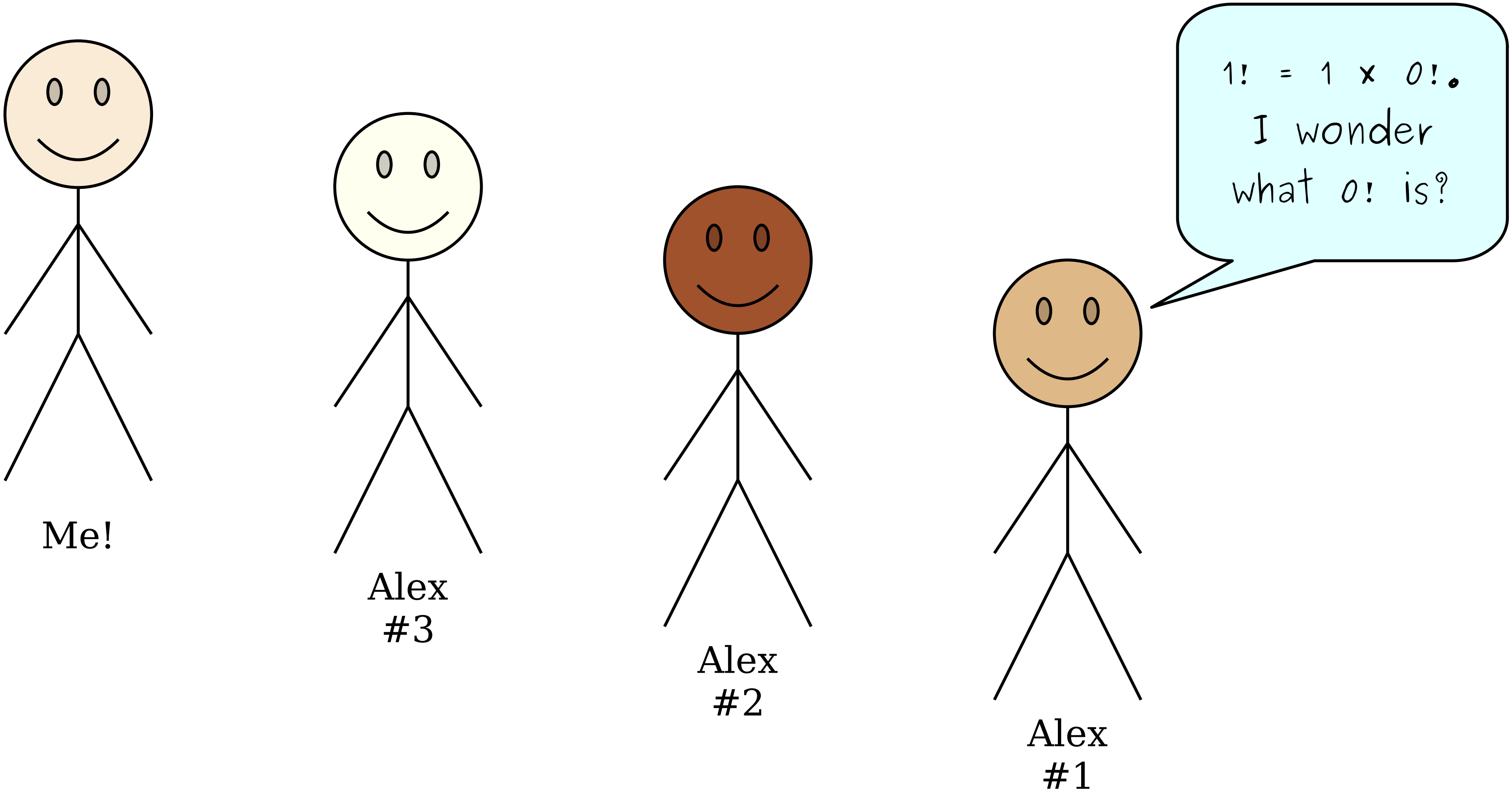
Alexes Compute Factorials



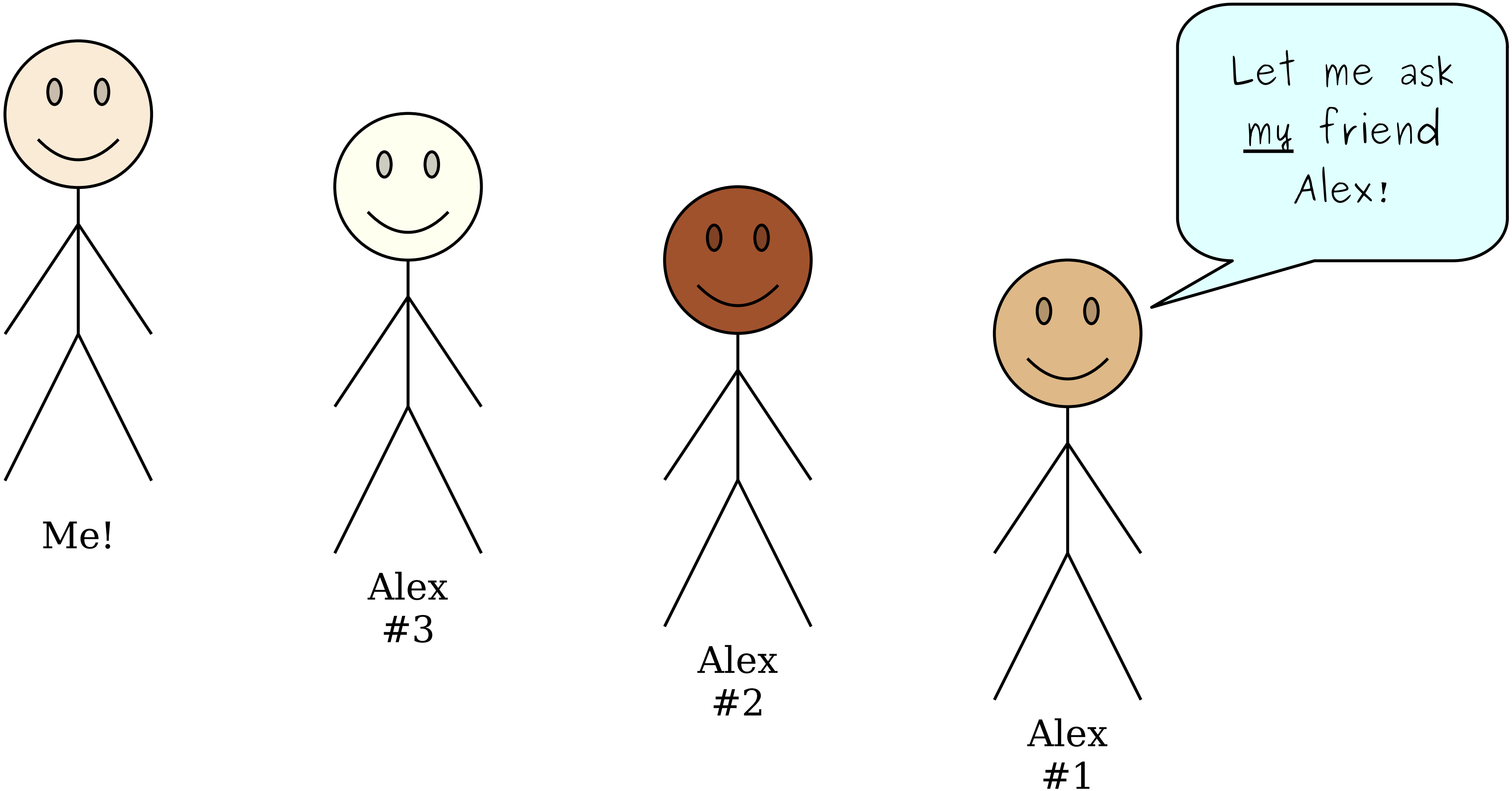
Alexes Compute Factorials



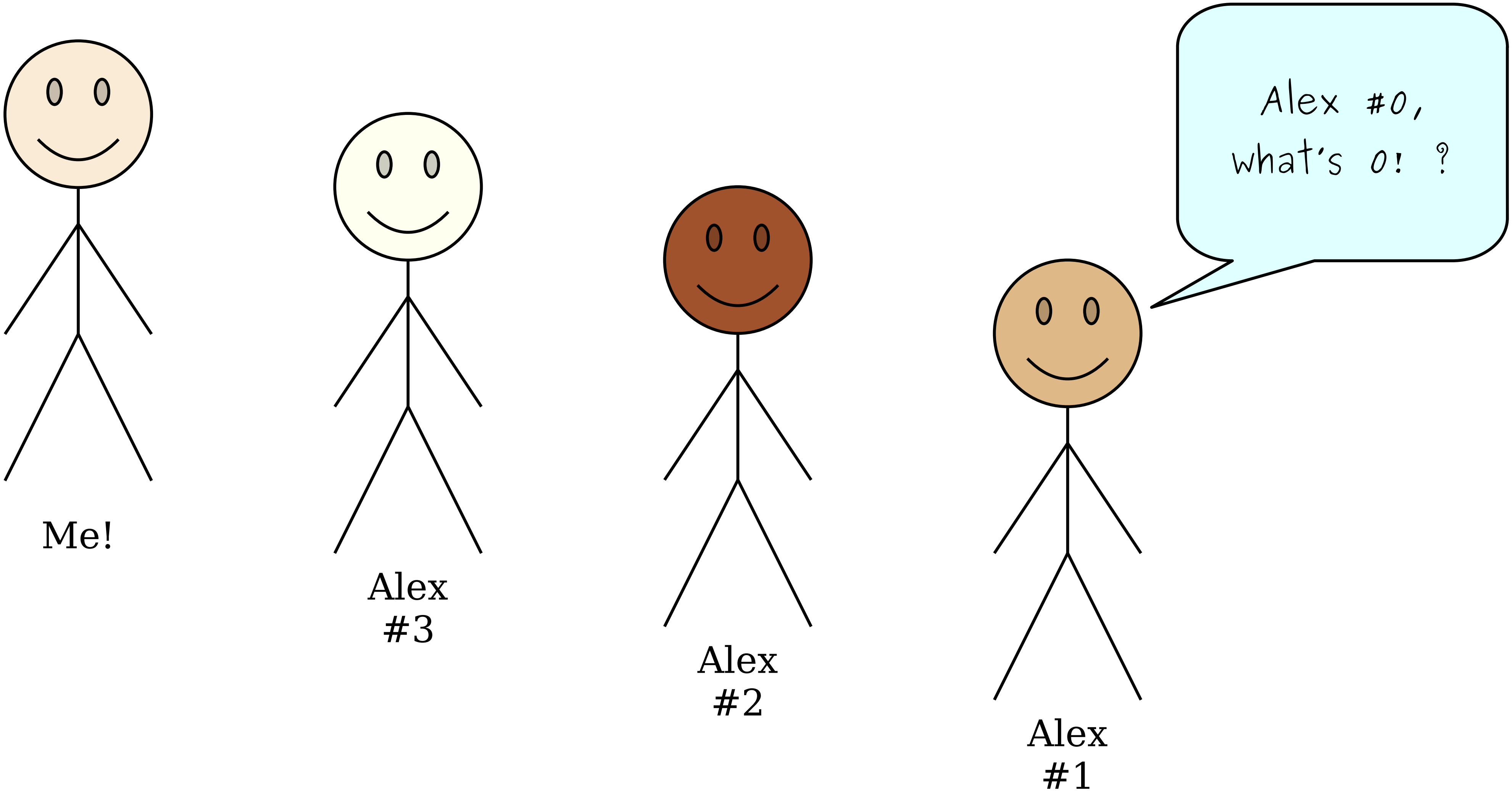
Alexes Compute Factorials



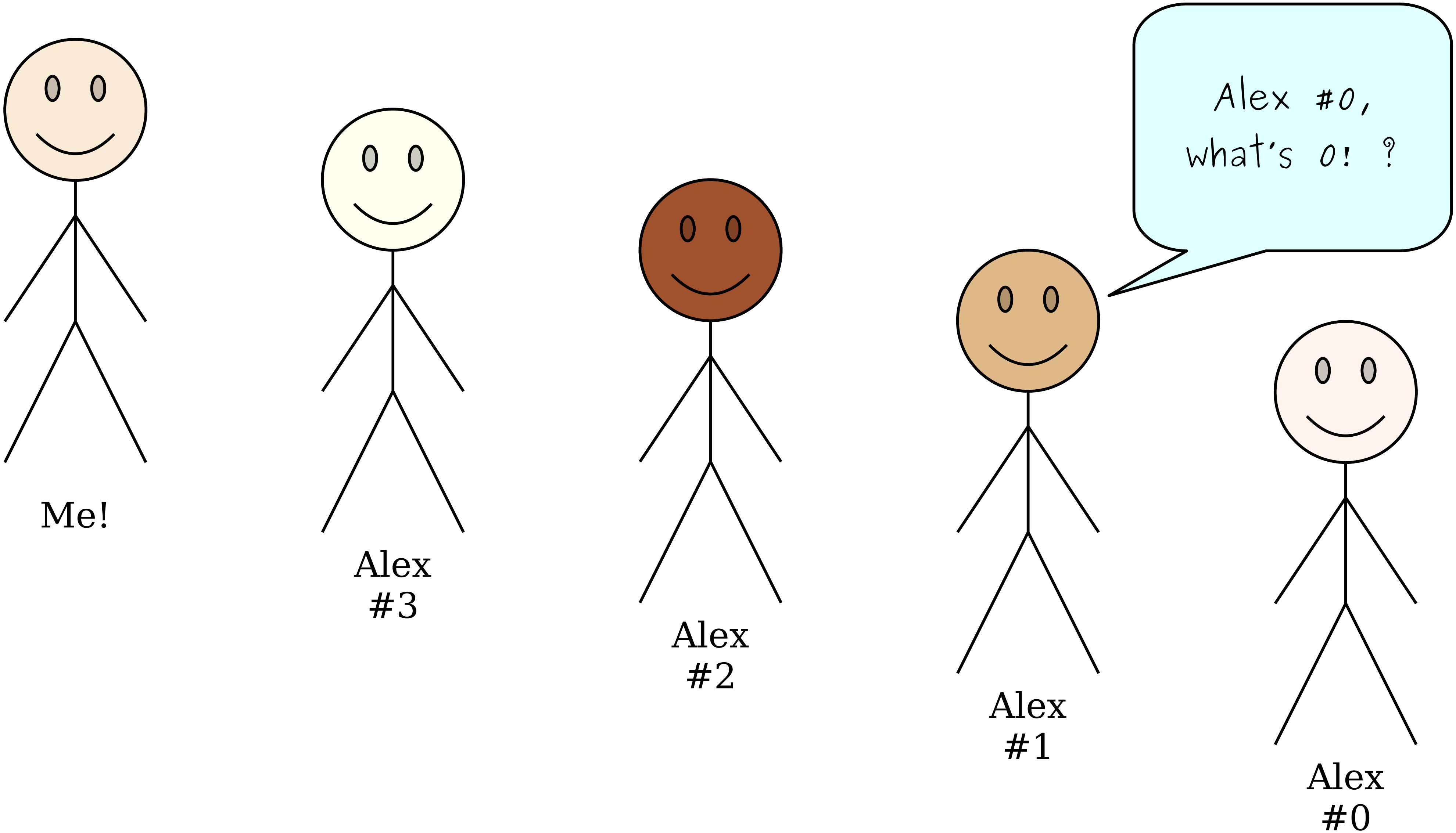
Alexes Compute Factorials



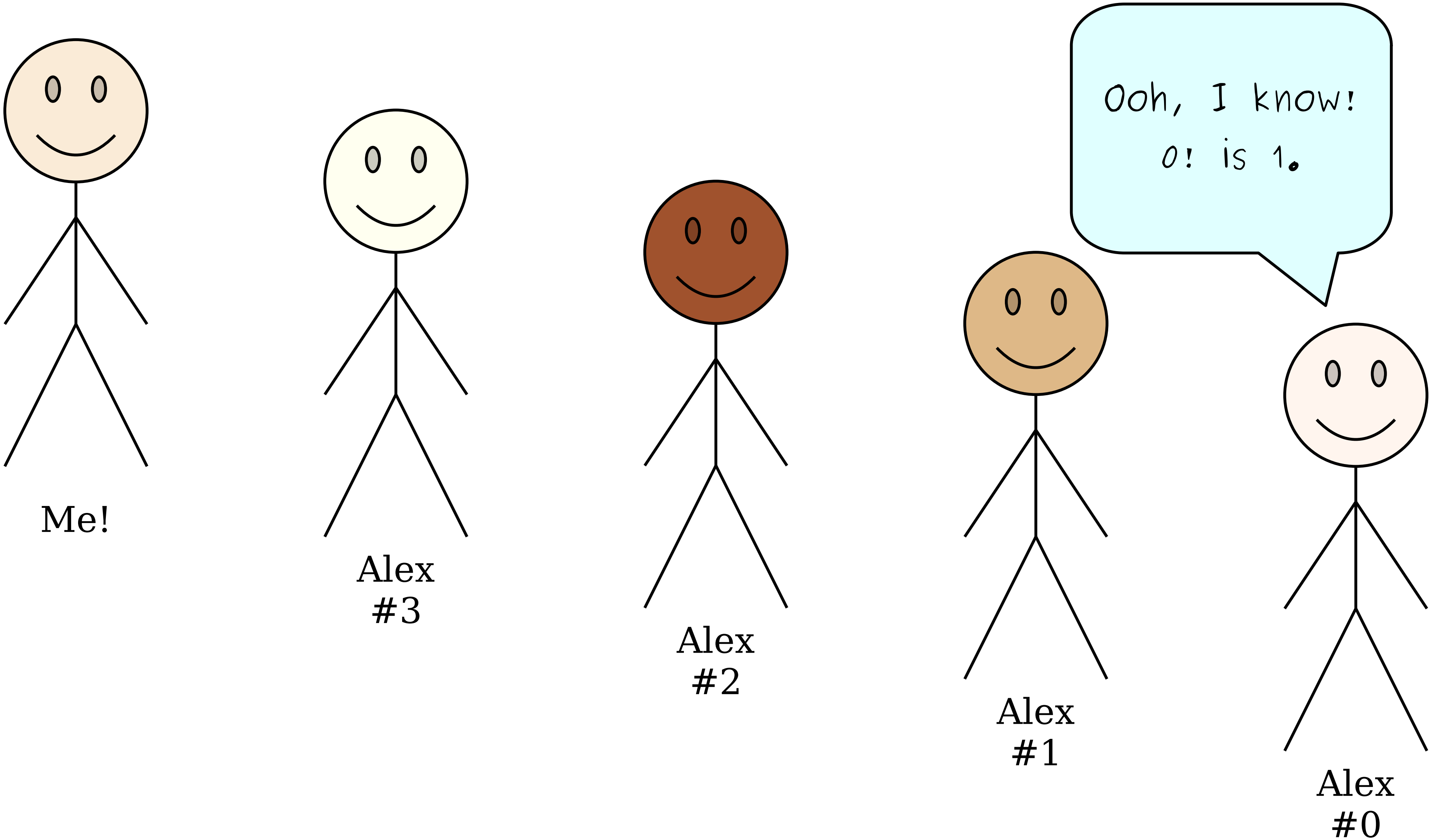
Alexes Compute Factorials



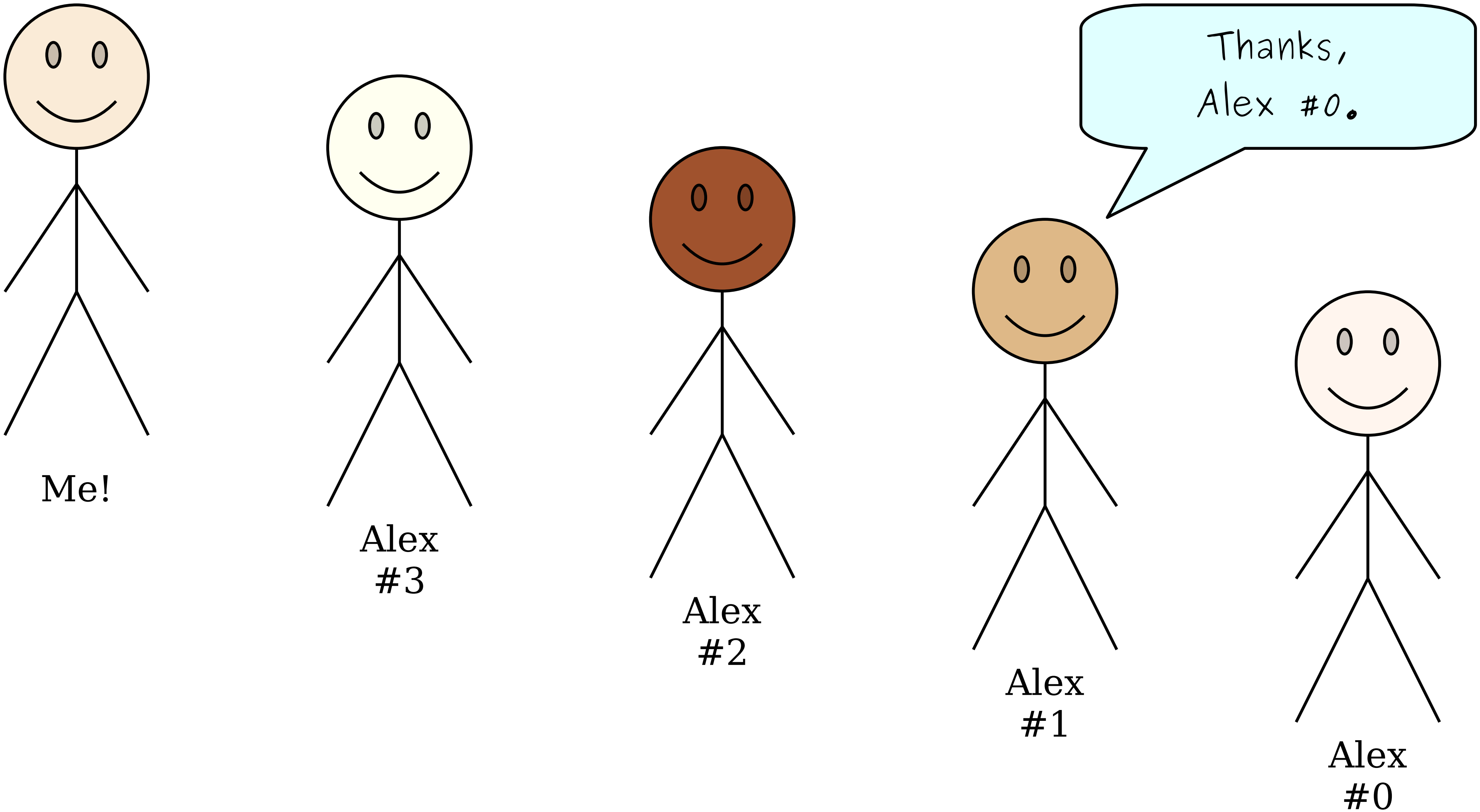
Alexes Compute Factorials



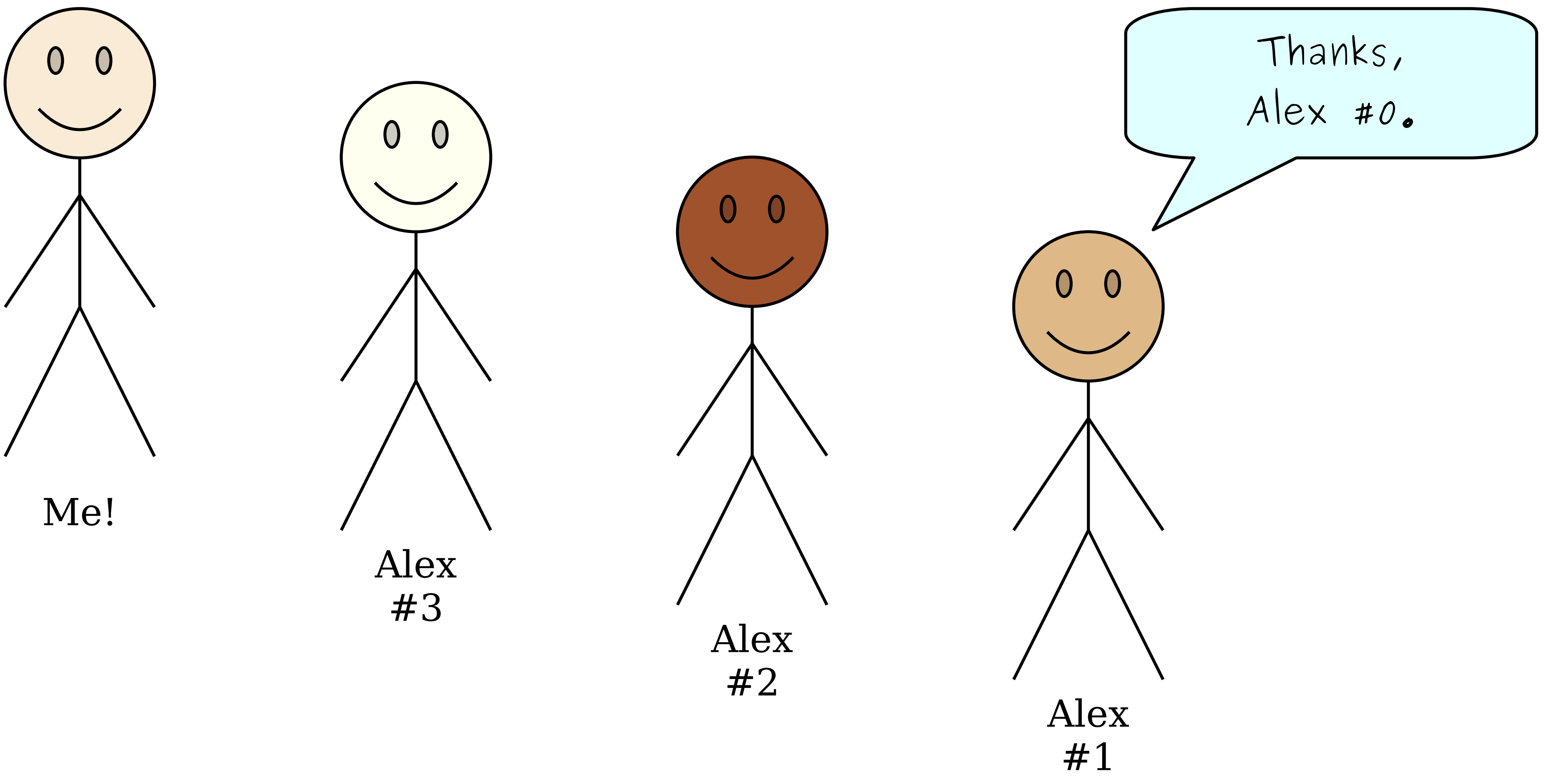
Alexes Compute Factorials



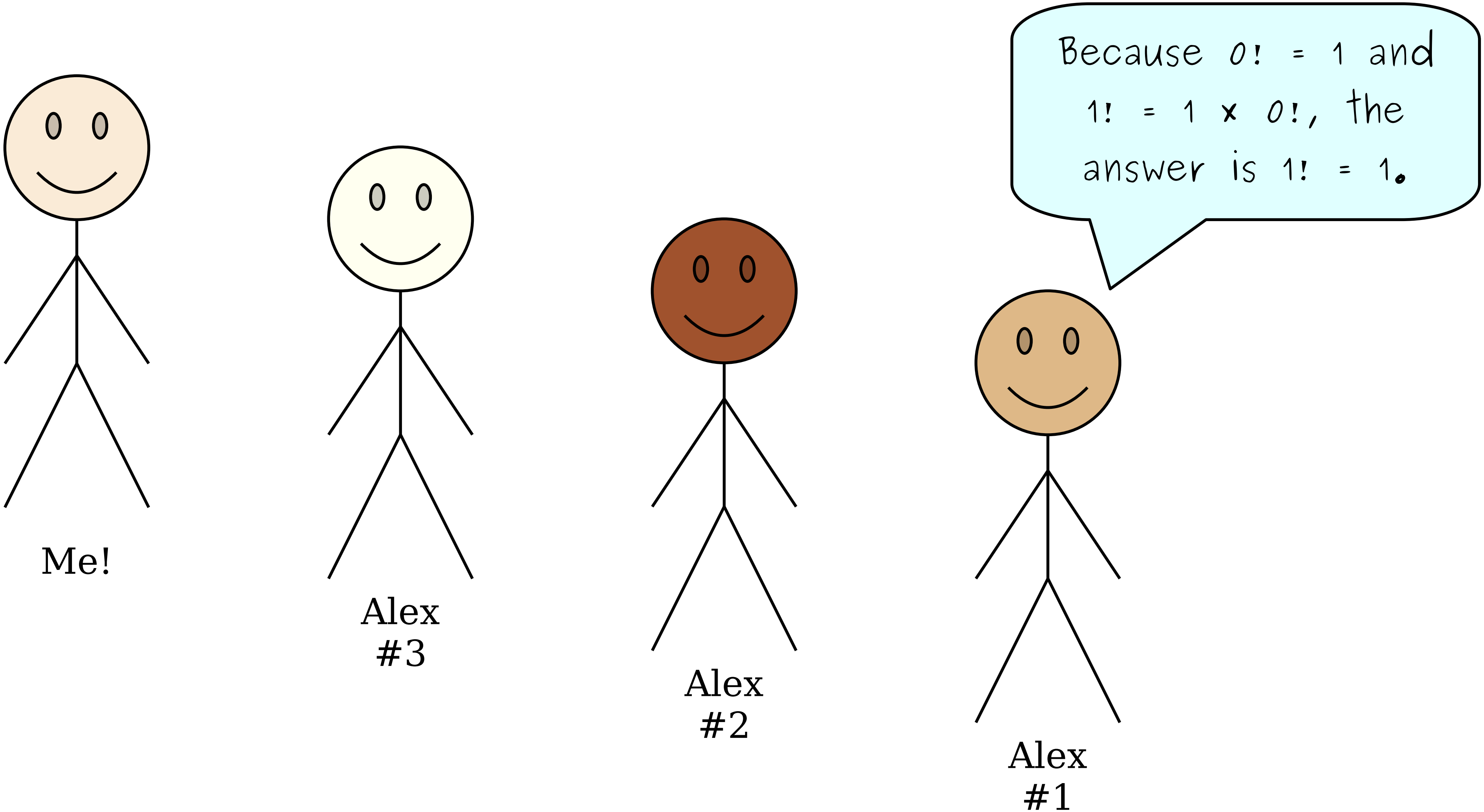
Alexes Compute Factorials



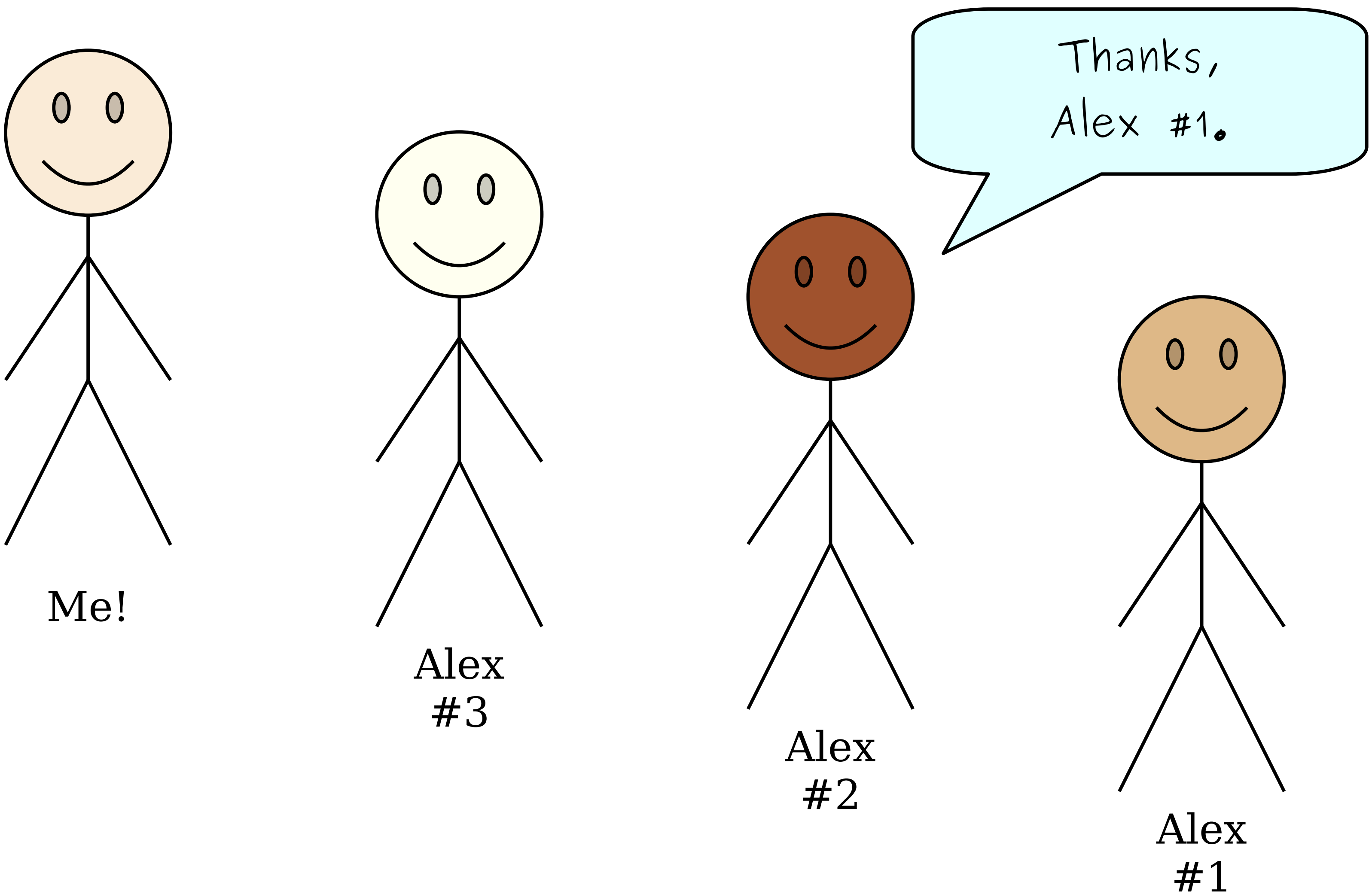
Alexes Compute Factorials



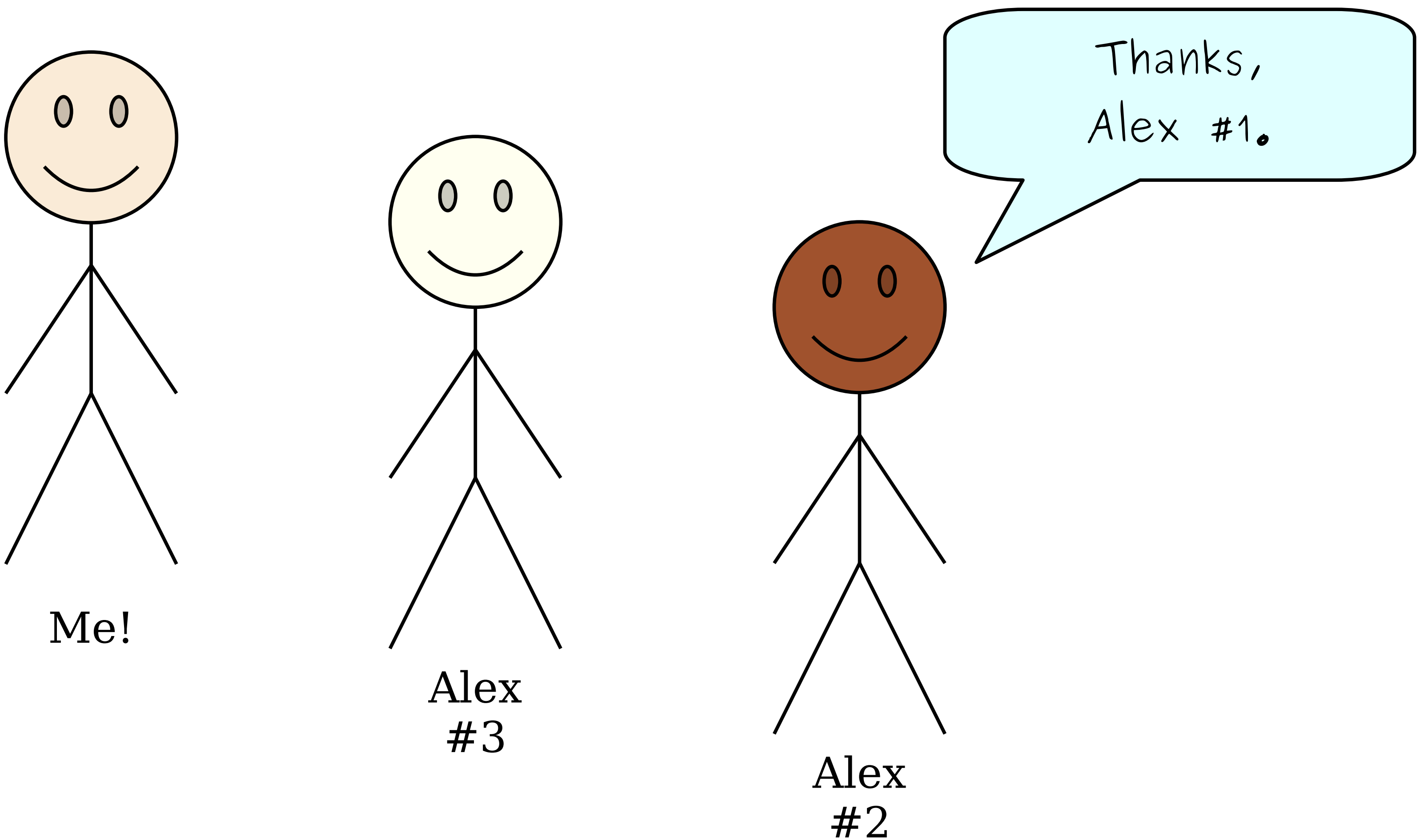
Alexes Compute Factorials



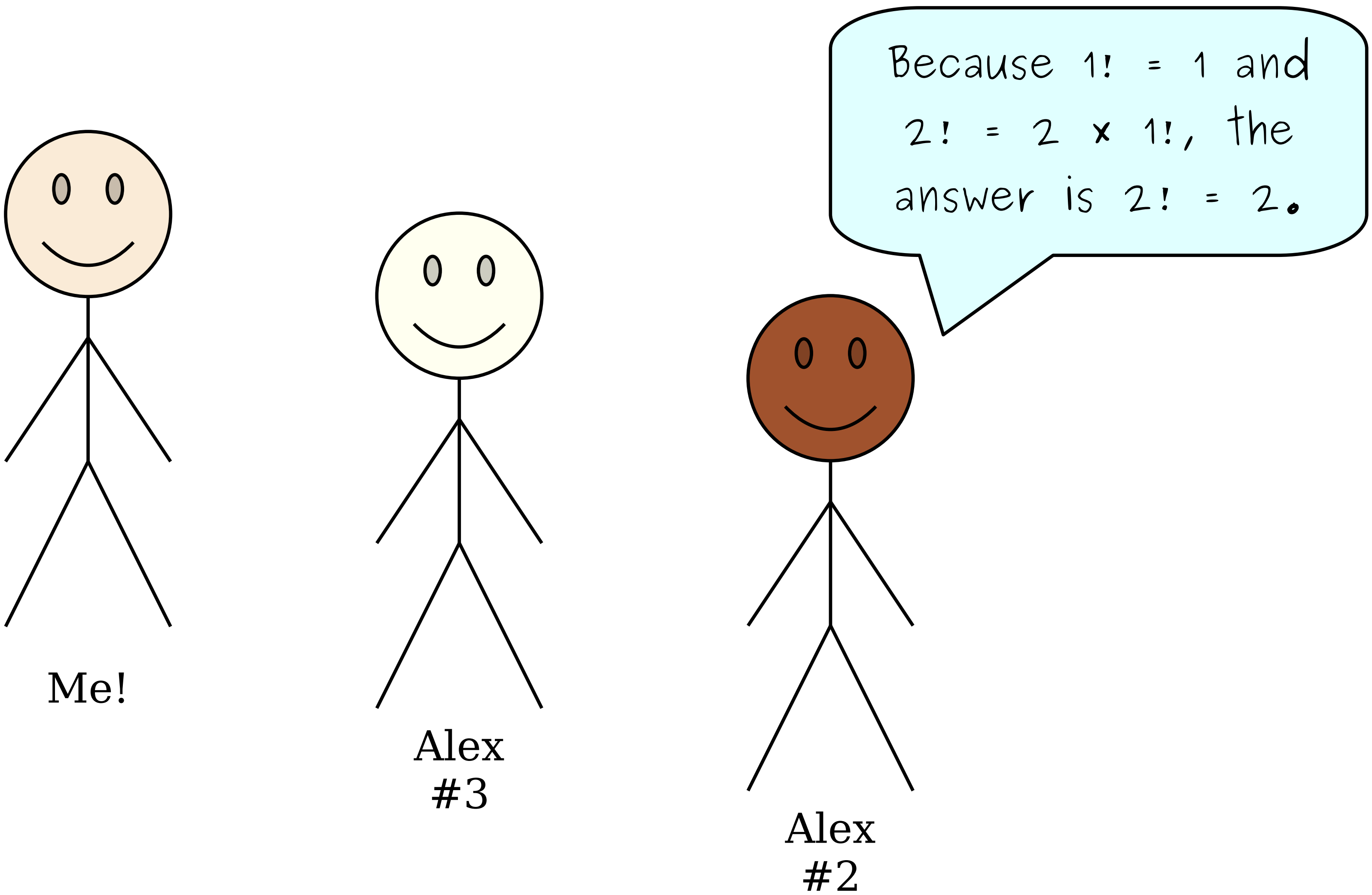
Alexes Compute Factorials



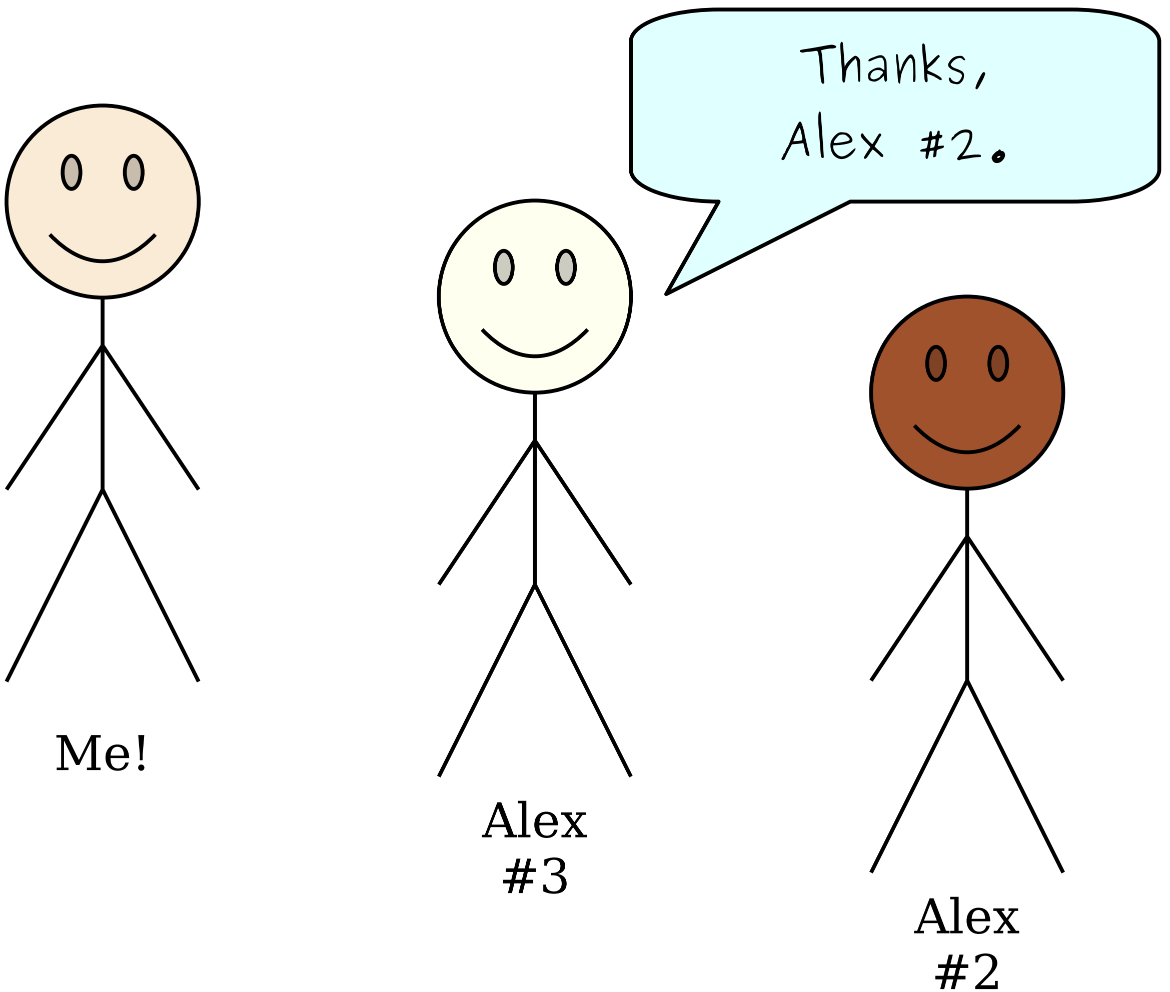
Alexes Compute Factorials



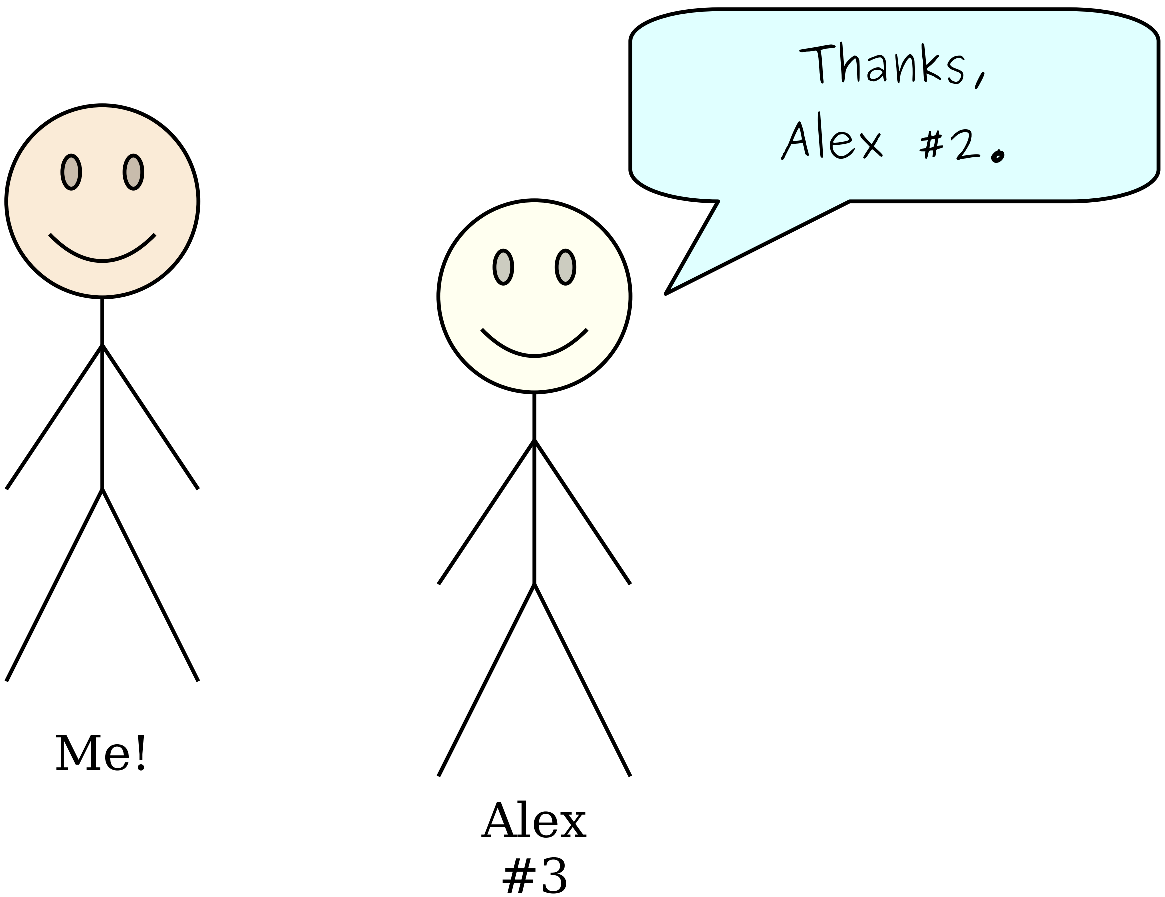
Alexes Compute Factorials



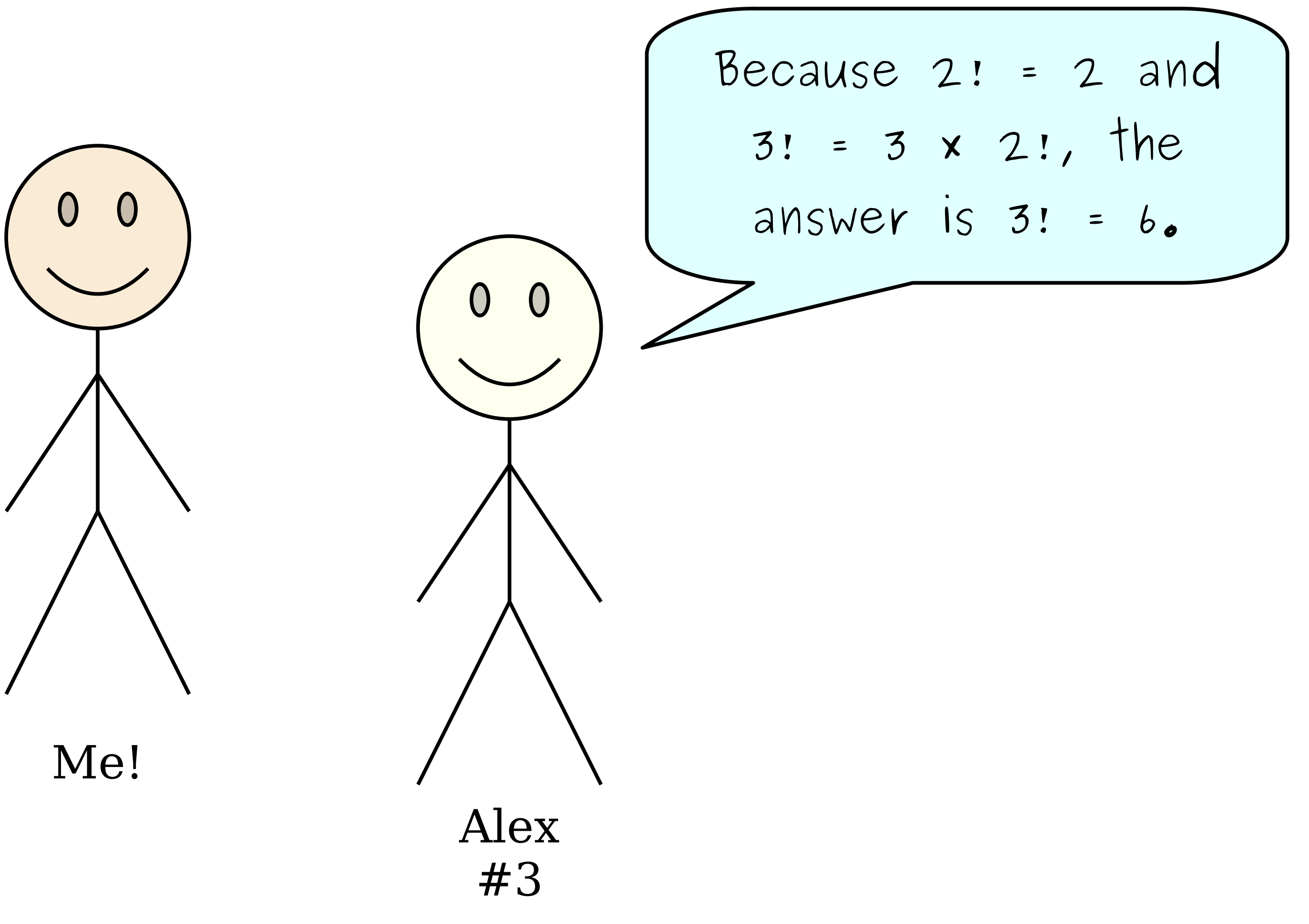
Alexes Compute Factorials



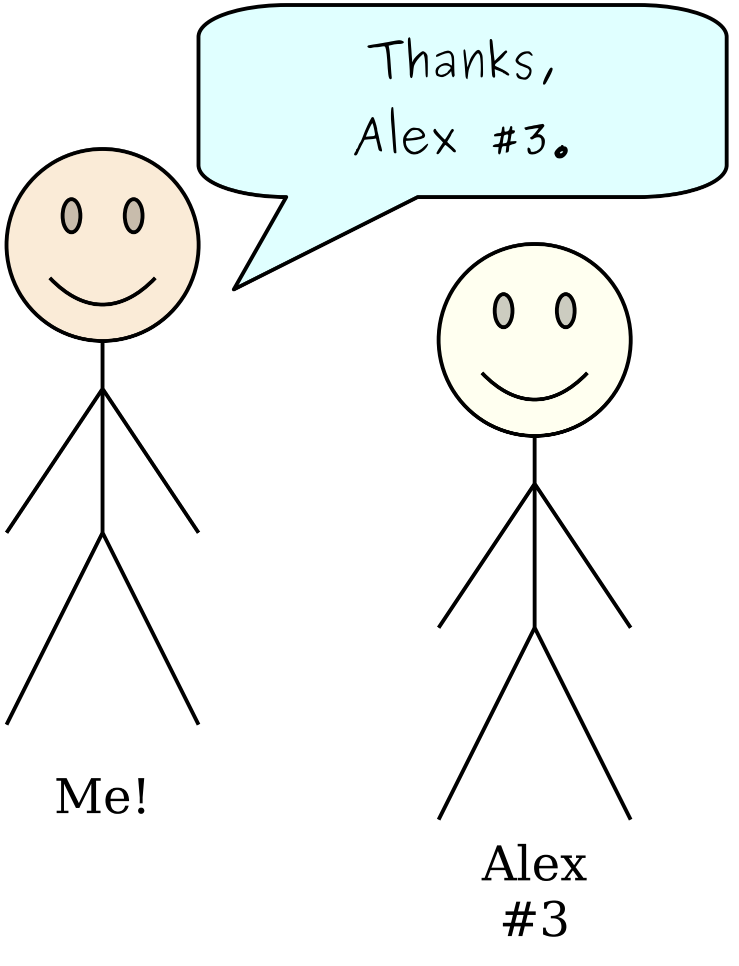
Alexes Compute Factorials



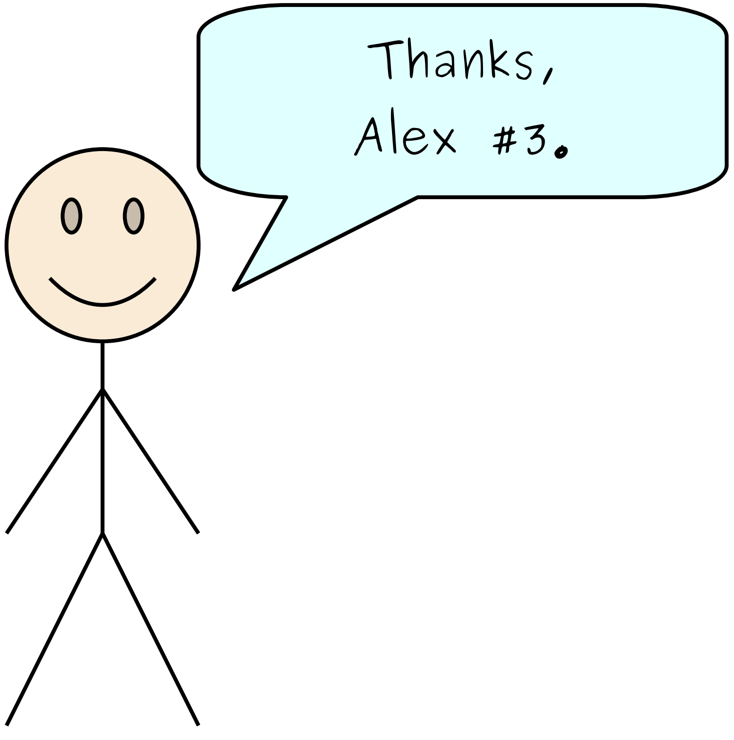
Alexes Compute Factorials



Alexes Compute Factorials

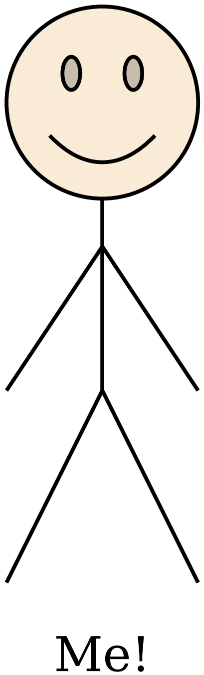


Alexes Compute Factorials



Me!

Alexes Compute Factorials



There are multiple people, each named Alex, but they’re not the same person.

Each Alex is tasked with computing a different number factorial.

Each Alex gives their answer back to the previous person.

Eventually I get the answer!

Recursion in Action

```
int main() {
    int nFact = factorial(3);
    cout << "3! = " << nFact << endl;

    return 0;
}
```

Recursion in Action

```
int main() {
    int nFact = factorial(3);
    cout << "3! = " << nFact << endl;

    return 0;
}
```

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3
int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

3

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

3

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

`int n`

Every time we call `factorial()`, we get a new copy of the local variable `n` that's independent of all the previous copies.

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2
int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

`int n`

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

1

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n: 1

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

0

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

0

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

0

int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

1

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1 1

int n: 1

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

1 1

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1 × 1

1

int n

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

1

int n

1

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2 1

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2 1

Recursion in Action

```
int main() {
```

```
int factorial(int n) {
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2 × 1

2
int n

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

2

int n

2

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

3

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

3 2

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

3 × 2

Recursion in Action

```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

3

int n

6

Recursion in Action

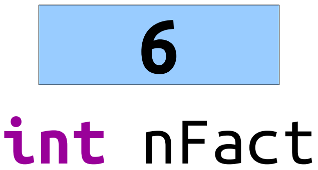
```
int main() {
    int nFact = factorial(3);
    cout << "3! = " << nFact << endl;

    return 0;
}
```

Recursion in Action

```
int main() {
    int nFact = factorial(3);
    cout << "3! = " << nFact << endl;

    return 0;
}
```



Thinking Recursively

- Solving a problem with recursion requires two steps.
- First, determine how to solve the problem for simple cases.
  - This is called the ***base case***.
- Second, determine how to break down larger cases into smaller instances.
  - This is called the ***recursive step***.

Summing Up Digits

- On Wednesday, we wrote this function to sum up the digits of a nonnegative integer:

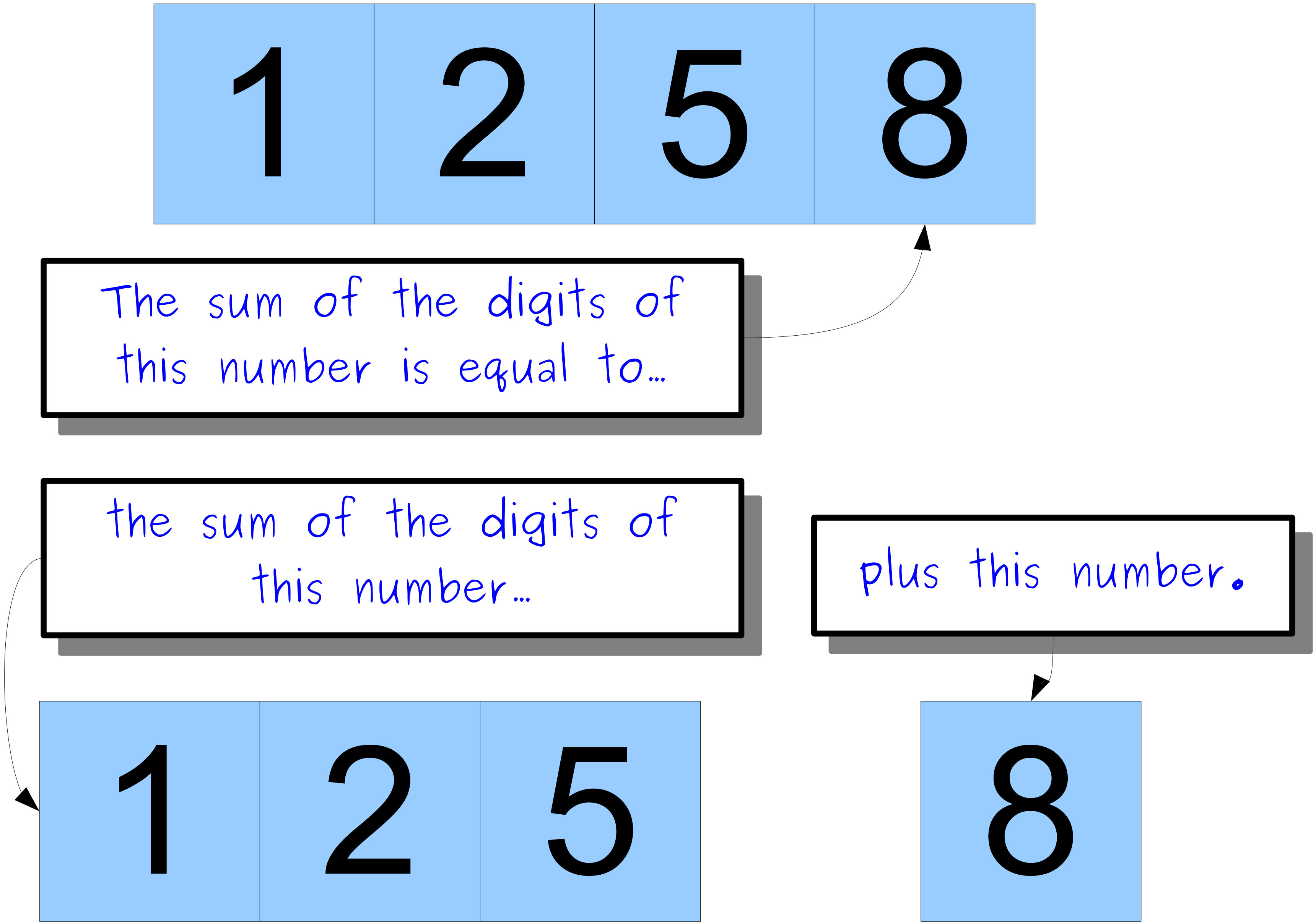
```
int sumOfDigitsOf(int n) {
    int result = 0;
    while (n > 0) {
        result += (n % 10);
        n /= 10;
    }
    return result;
}
```

- Let’s rewrite this function recursively!

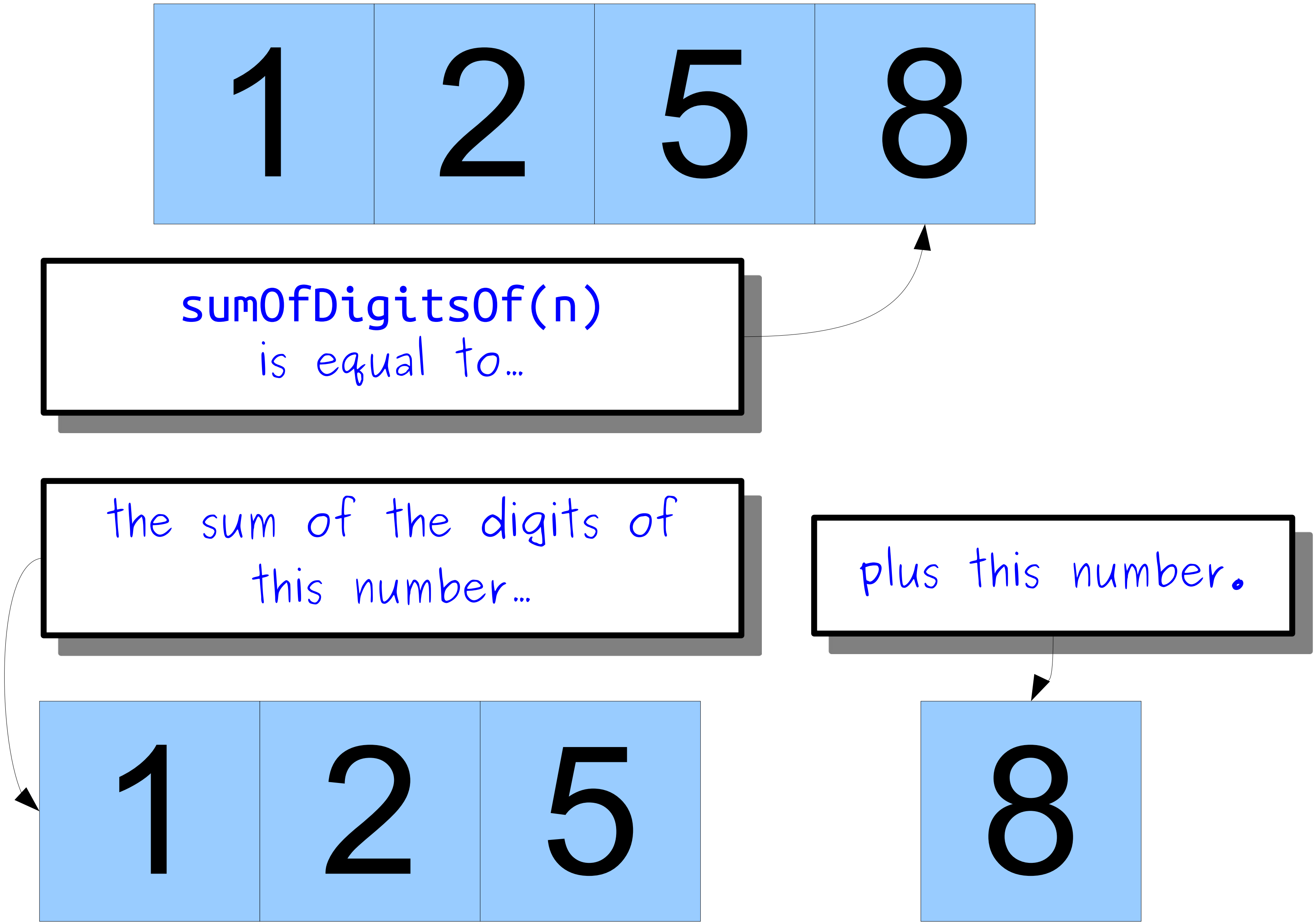
Summing Up Digits

- To write a recursive function, we need to think of a ***base case*** and a ***recursive case***.
- The ***base case*** produces answers when the input is sufficiently simple.
- The ***recursive case*** takes more complex inputs and simplifies them, taking them closer to the base case.
- What's a reasonable base case for our sum of digits function?

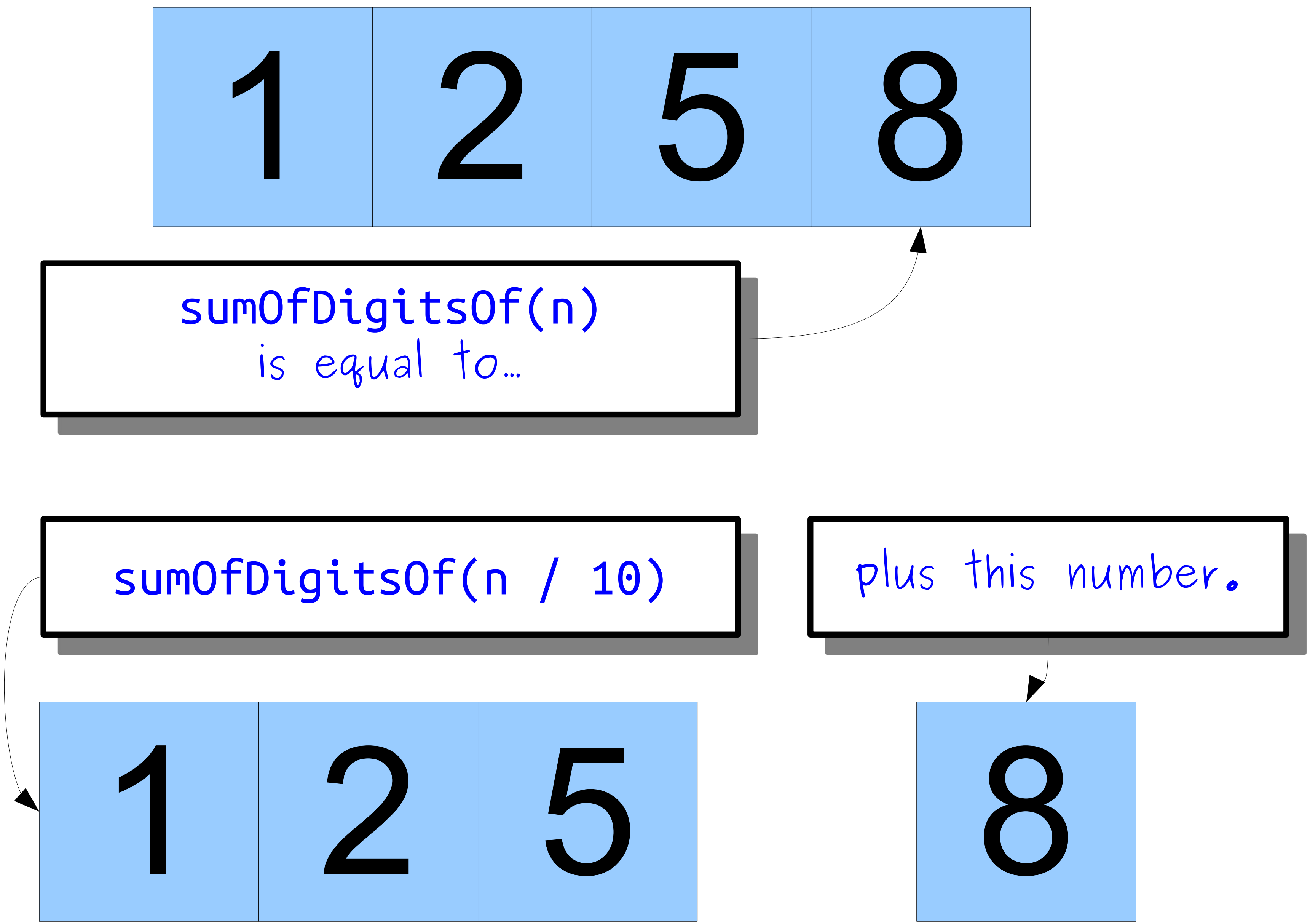
Summing Up Digits



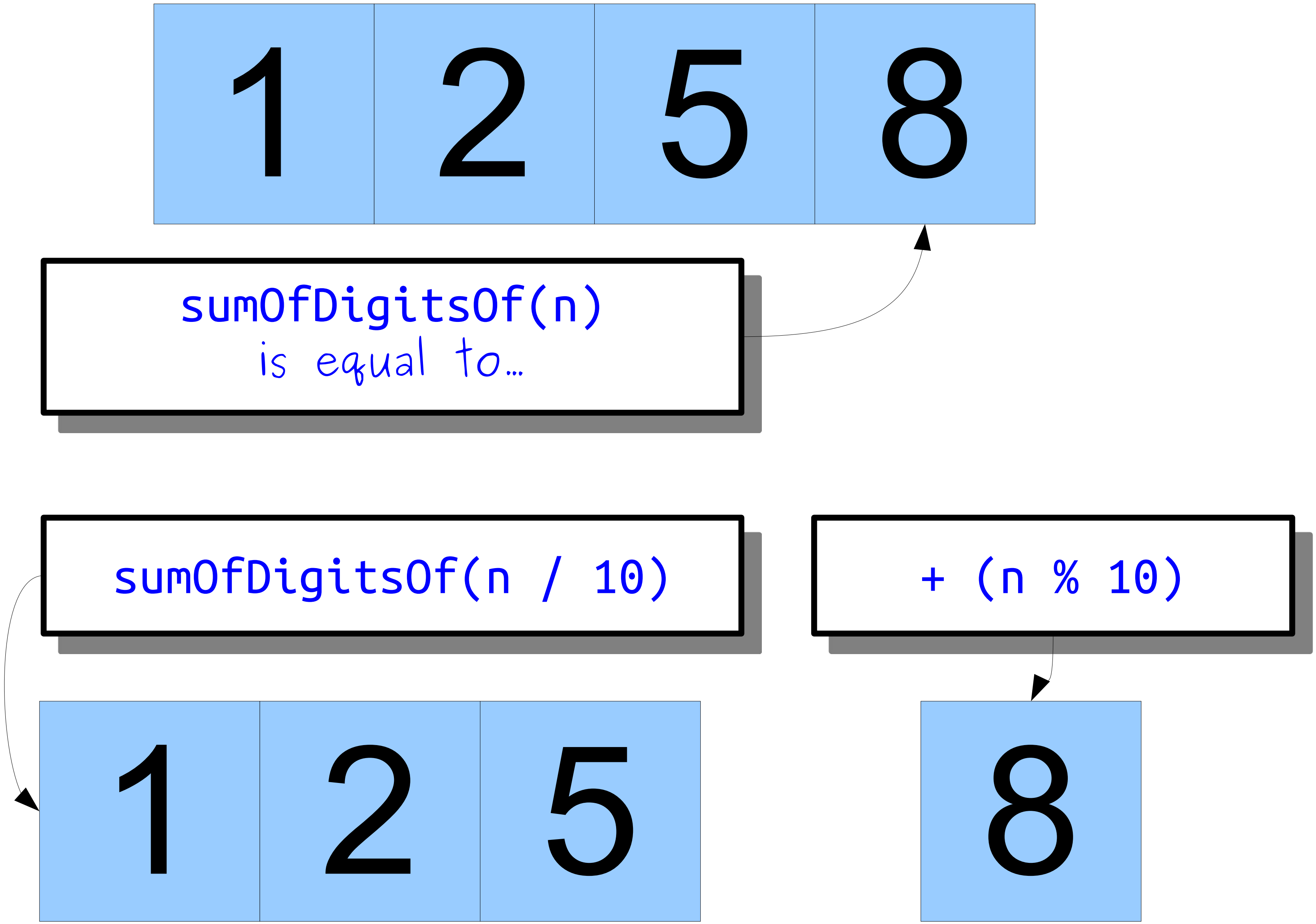
Summing Up Digits



Summing Up Digits



Summing Up Digits



Tracing the Recursion

```
int main() {
   int sum = sumOfDigitsOf(137);
   cout << "Sum is " << sum << endl;
}
```

Tracing the Recursion

```
int main() {
    int sum = sumOfDigitsOf(137);
    cout << "Sum is " << sum << endl;
}
```

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n 137

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n 137

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 137

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 137

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n 137

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {

}
```

```
int sumOfDigitsOf(int n) {                    int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 13

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                          int n [ 13 ]
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 13

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 13

Tracing the Recursion

```
int main() {
```

```
int sumOfDigitsOf(int n) {
```

```
int sumOfDigitsOf(int n) {
```

```
int sumOfDigitsOf(int n) {                                   int n [ 1 ]
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 1

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 1

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {              int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }                   1
}
```

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n 13

1

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 13

1 + 3

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 13

4

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 137

4

Tracing the Recursion

```
int main() {
```

```
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n 137

4

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 137

4 + 7

Tracing the Recursion

```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

int n: 137

11

Tracing the Recursion

```
int main() {
    int sum = sumOfDigitsOf(137);
    cout << "Sum is " << sum << endl;
}
```

11

Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem into one or more
    smaller problems with the same
    structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall
    solution.
    Return the overall solution.
}
```

These simple cases are called *base cases*.

These are the *recursive cases*.

Time-Out for Announcements!

Outdoor Activities Guide

- If case you're looking for things to do in the area this weekend, I've posted an Outdoor Activities Guide on the course website.
- It's a mix of places to go and places to get a bite to eat.
- Some highlights:
  - See the whole Santa Clara Valley and beyond from the observatory on Mt. Hamilton.
  - Walk among giant redwood trees and pick your own bay leaves.
  - Catch a gorgeous sunset view of San Francisco from an artificial island covered in guerrilla artwork.
  - Get cheap, delicious food from restaurants tucked into unassuming strip malls.
- Enjoy!

Section Signups

- Section signups are open!
- Sign up for section at

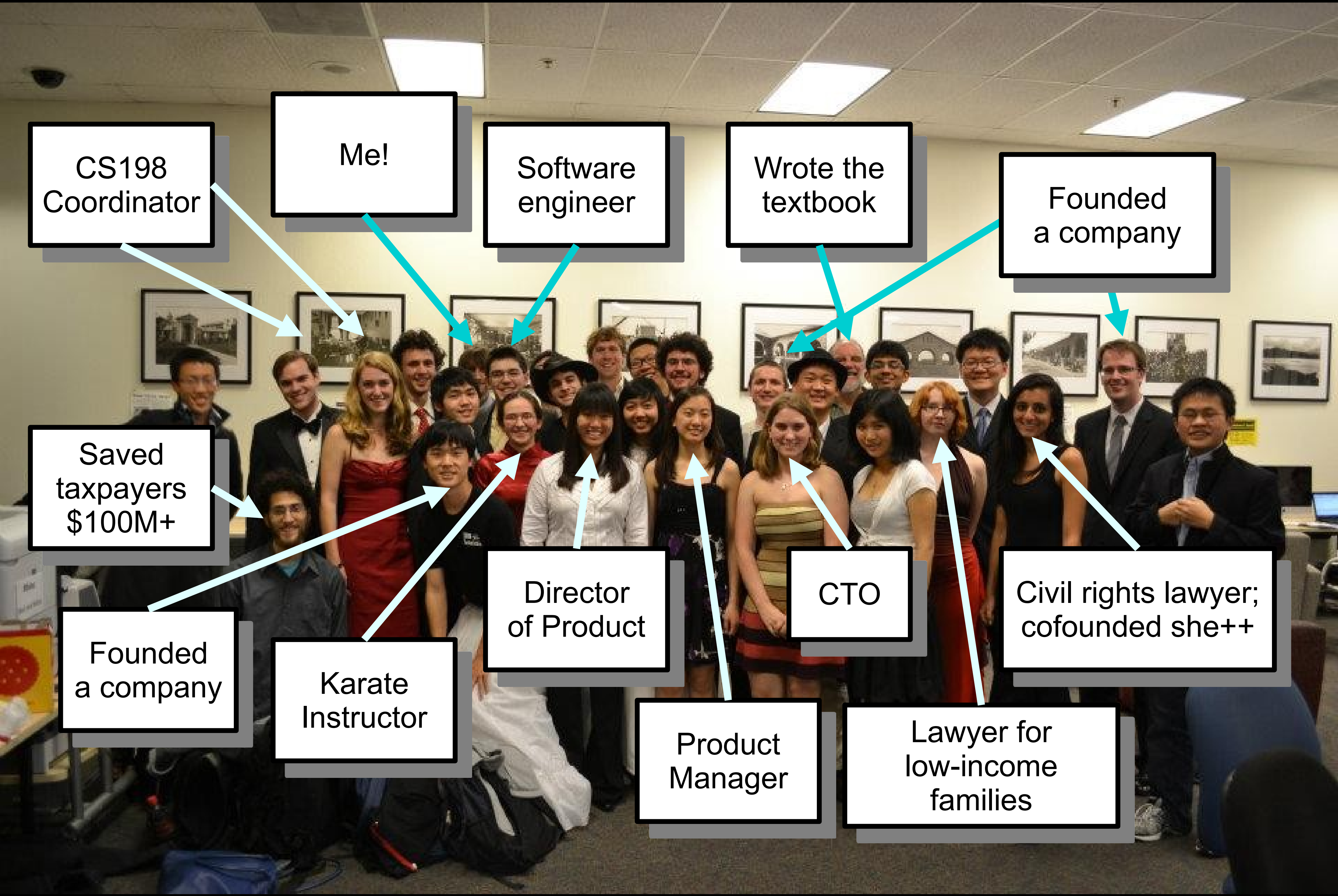
  https://cs198.stanford.edu/cs198/auth/default.aspx

  by Sunday at 5PM.
- Reminders:
  - We don't look at Axess when determining discussion sections. You still need to sign up here even if you have a section on Axess.
  - Courses like CS106L, CS106BACE, and CS106S are taken *in addition to* discussion sections rather than *in place of* sections.
  - If you miss the Sunday 5PM deadline, signups reopen on Tuesday on a first-come-first-served basis.
- Sections start next week.

Assignment 1

- Assignment 0 was due today at 1:00PM Pacific.
- ***Assignment 1: Welcome to C++*** goes out today. It's due on Friday, January 17th at 1:00PM Pacific.
  - Play around with C++ and the Stanford libraries!
  - Get some practice with recursion!
  - Explore the debugger!
  - See some pretty pictures!
- We recommend making slow and steady progress on this assignment throughout the course of the week. There's a recommended timetable at the top of the assignment description.

Getting Help


CS198 Coordinator
Me!
Software engineer
Wrote the textbook
Founded a company
Saved taxpayers $100M+
Founded a company
Karate Instructor
Director of Product
Product Manager
CTO
Lawyer for low-income families
Civil rights lawyer; cofounded she++

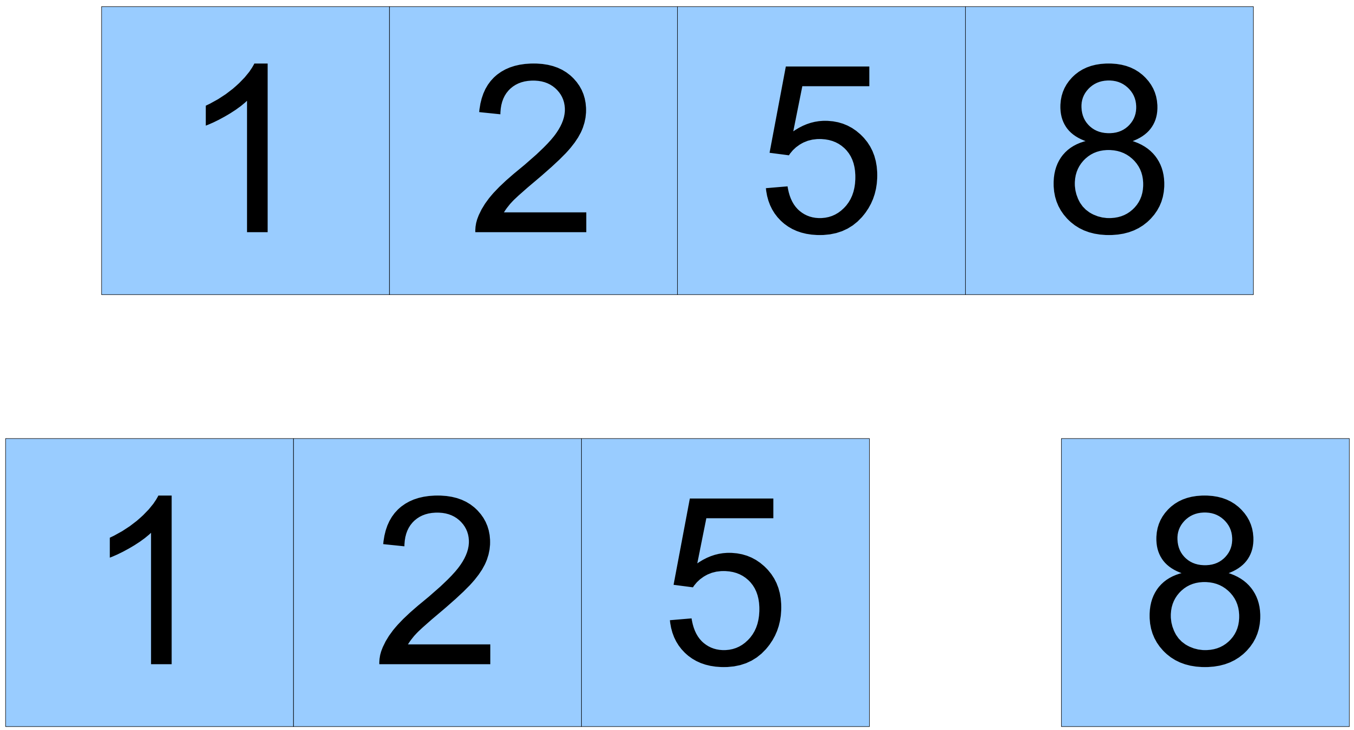
Getting Help

- ***LaIR Hours***
  - Sunday – Thursday, 7PM – 11PM Pacific.
  - Starts Sunday.
  - Runs in the Durand building 3rd floor.
- ***Jonathan's and Keith's Office Hours***
  - Check the website for times and places.

One More Unto the Breach!

Recursion and Strings

Thinking Recursively



Thinking Recursively

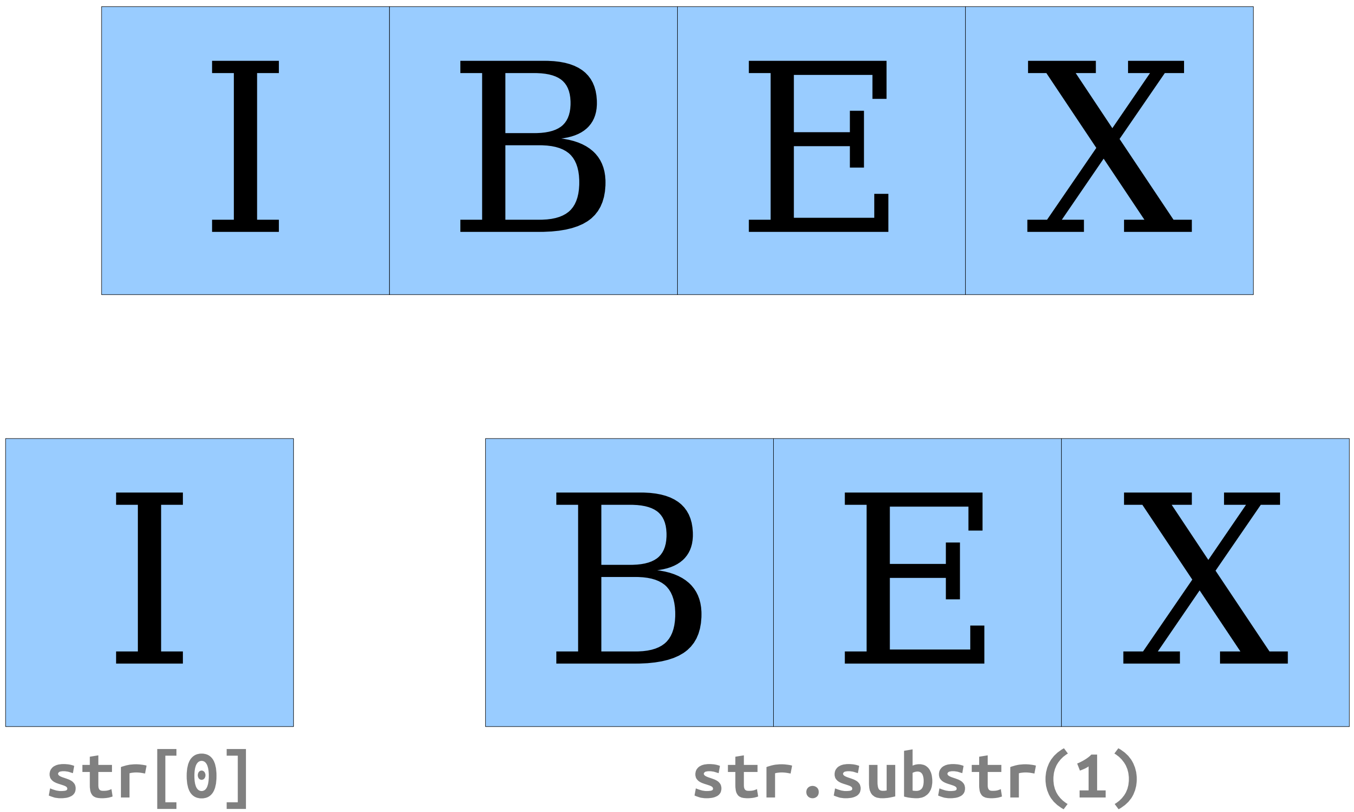
I B E X

I B E X

`str[0]` ???

Answer at
https://cs106b.stanford.edu/pollev

Thinking Recursively



How do you reverse a string?
?gnirts a esrever uoy od woH

Reversing a String

N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

Reversing a String

N	u	b	i	a	n		I	b	e	x

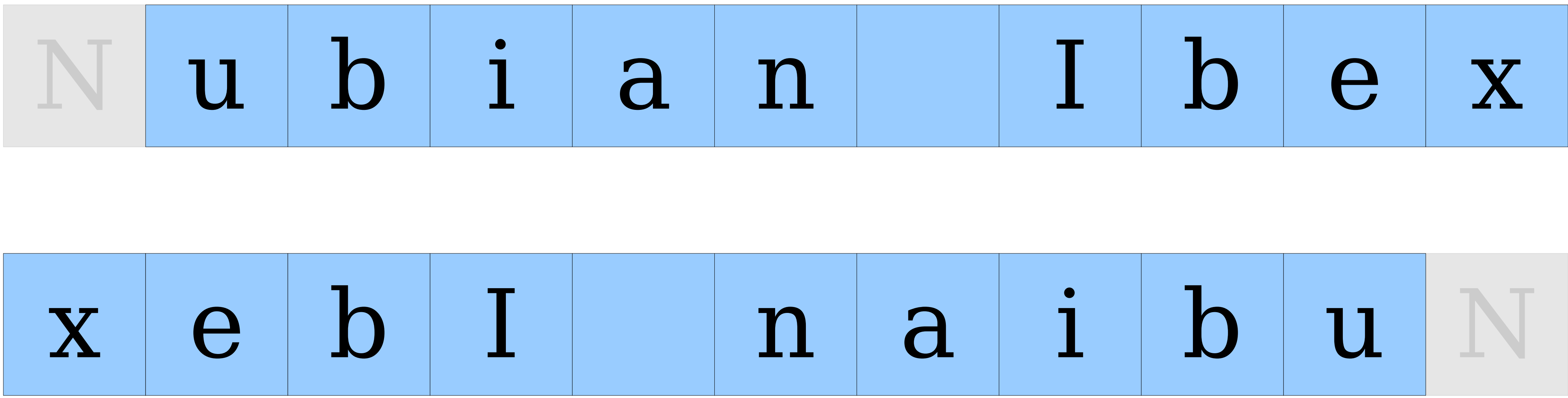
x	e	b	I		n	a	i	b	u	N

Reversing a String

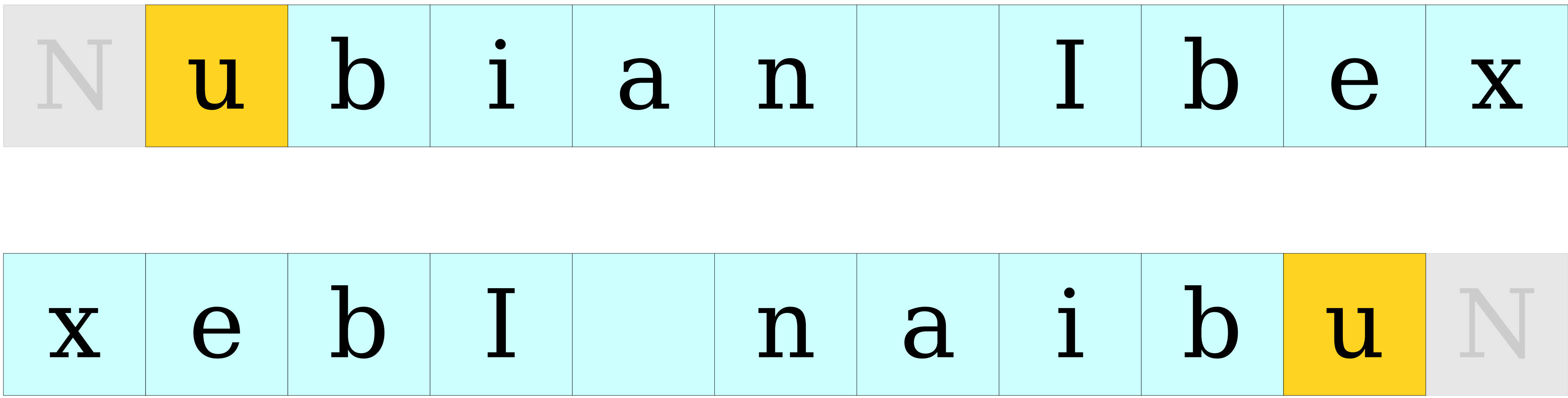
N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

Reversing a String



Reversing a String



Reversing a String Recursively

Reversing a String Recursively

`reverseOf("` T O P `") =`

Reversing a String Recursively

reverseOf(" T O P ") = reverseOf(" O P ") + T

Reversing a String Recursively

reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") =

Reversing a String Recursively

reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") = reverseOf(" P ") + O

Reversing a String Recursively

reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") = reverseOf(" P ") + O

reverseOf(" P ") =

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P") =   reverseOf("") + P
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") = reverseOf("P") + O

reverseOf("P") = reverseOf("") + P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T
reverseOf("OP") =  reverseOf("P") + O
reverseOf("P") =   "" + P
reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") =   reverseOf(" P ") + O

reverseOf(" P ") =             P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") =                 P + O

reverseOf(" P ") =                   P

reverseOf("") = ""
```

Reversing a String Recursively

reverseOf(" T O P ") = reverseOf(" O P ") + T

reverseOf(" O P ") = P O

reverseOf(" P ") = P

reverseOf("") = ""

Reversing a String Recursively

reverseOf("TOP") = PO + T

reverseOf("OP") = PO

reverseOf("P") = P

reverseOf("") = ""

Reversing a String Recursively

```
reverseOf("TOP") =    POT

reverseOf("OP") =     PO

reverseOf("P") =      P

reverseOf("") = ""
```

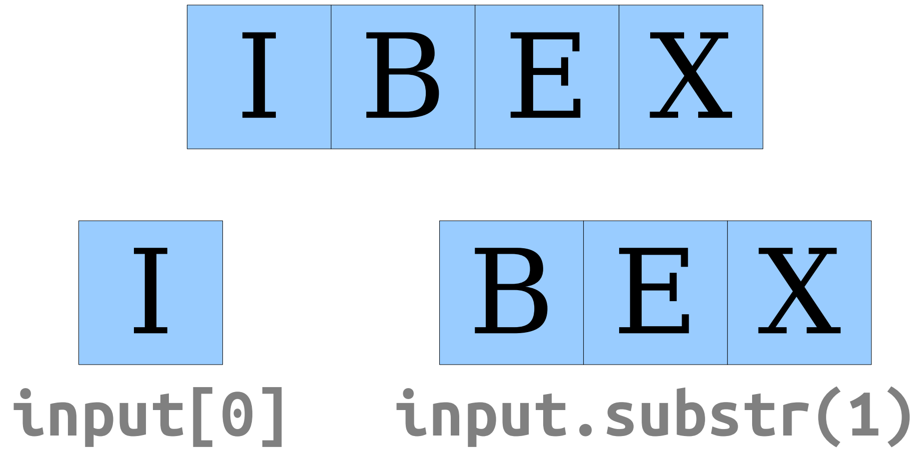
Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P") =   reverseOf("") + P

reverseOf("") = ""
```



Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem into one or more
    smaller problems with the same
    structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall
    solution.
    Return the overall solution.
}
```

These simple cases are called *base cases*.

These are the *recursive cases*.

Recap from Today

- Recursion works by identifying
  - one or more ***base cases***, simple cases that can be solved directly, and
  - one or more ***recursive cases***, where a larger problem is turned into a smaller one.
- Recursion is everywhere! And you can use it on strings.

Your Action Items

- ***Sign Up for a Discussion Section***
  - Signups close this Sunday. Use the link we've shared rather than signing up on Axess.
- ***Read Chapter 7.***
  - This chapter is all about recursion.
- ***Start Working on Assignment 1.***
  - Aim to complete the Debugger Warmups by Monday and start working on Fire.

Next Time

- ***Reference Parameters***
  - On master copies and xeroxes.
- ***Vector***
  - Representing sequences.
- ***Recursion on Vectors***
  - Of course.