

Getting Started with Recursion

Logistics: PolleEV

Lecture Participation

- Starting next Wednesday, we will be using the website PollEV to ask questions in lecture.
- If you answer these questions in lecture – regardless of whether you're correct – you'll get attendance credit for the day.
- If you don't attend lecture in person, no worries! You can answer some online questions within 48 hours of the lecture to earn participation credit.

Lecture Participation

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

What is your favorite book?

- Answer online by visiting

<https://pollev.com/cs106bwin23/>

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

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

- 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!

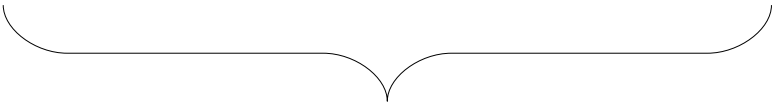
Computing Factorials

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

Computing Factorials

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

Computing Factorials

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

$$4!$$

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

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

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



$$3!$$

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

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

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

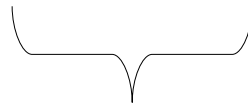
$$3! = 3 \times 2 \times 1$$

Computing Factorials

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

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

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



$2!$

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

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

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

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

$$2! = 2 \times 1$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

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

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

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

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

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

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

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

$$3! = 3 \times 2$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

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

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

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

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

$$4! = 4 \times 6$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

$$4! = 24$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

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

$$4! = 24$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

Computing Factorials

$$5! = 5 \times 24$$

$$4! = 24$$

$$3! = 6$$

$$2! = 2$$

$$1! = 1$$

$$0! = 1$$

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

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

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

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

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

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

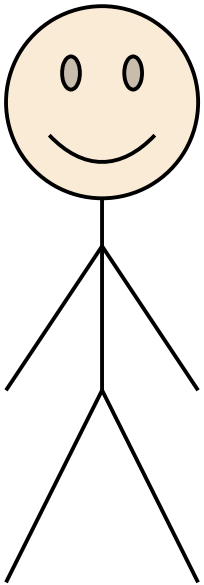
$$0! = 1$$

Another View of Factorials

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

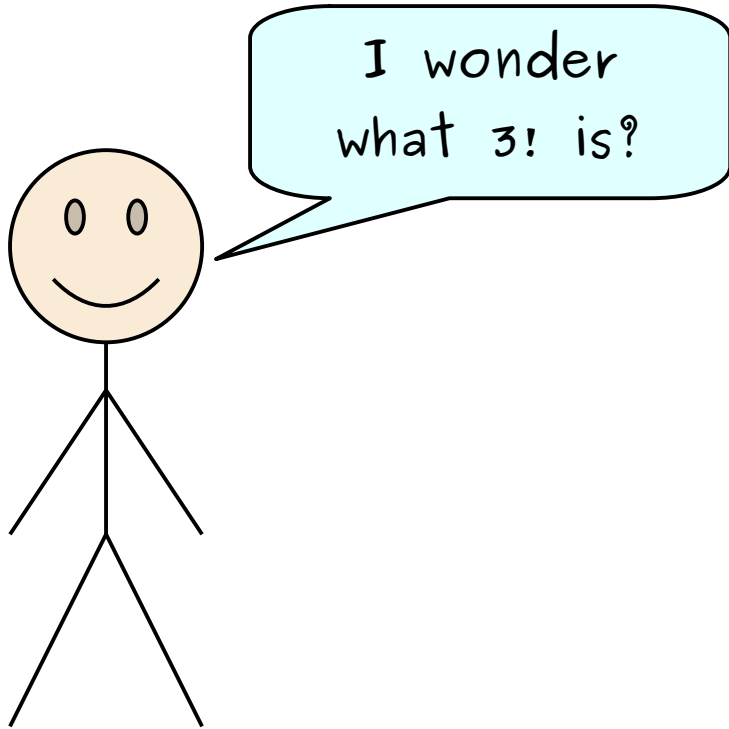
Andrews Compute Factorials

Andrews Compute Factorials



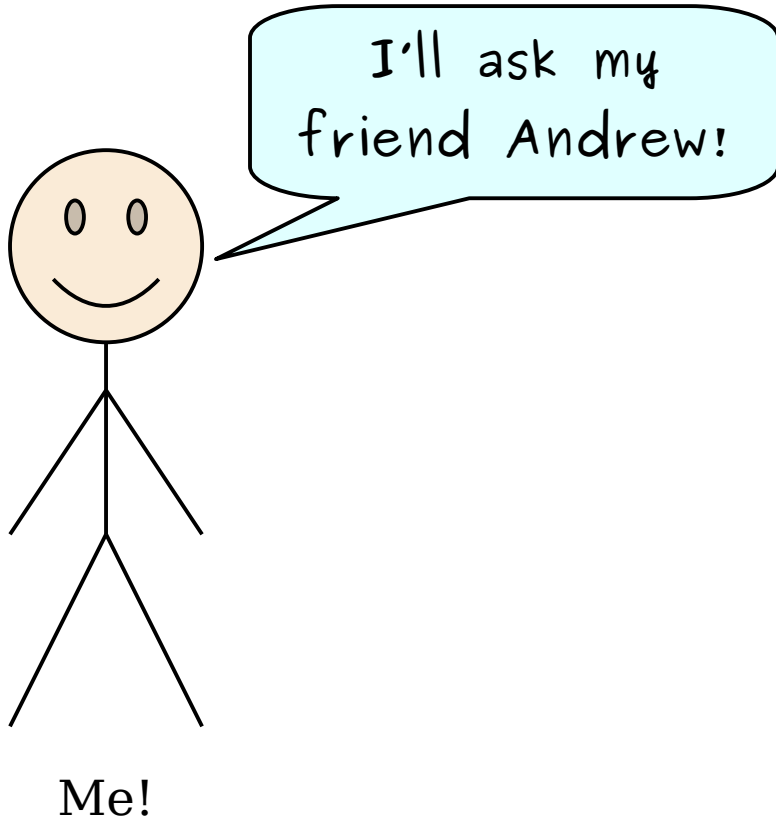
Me!

Andrews Compute Factorials

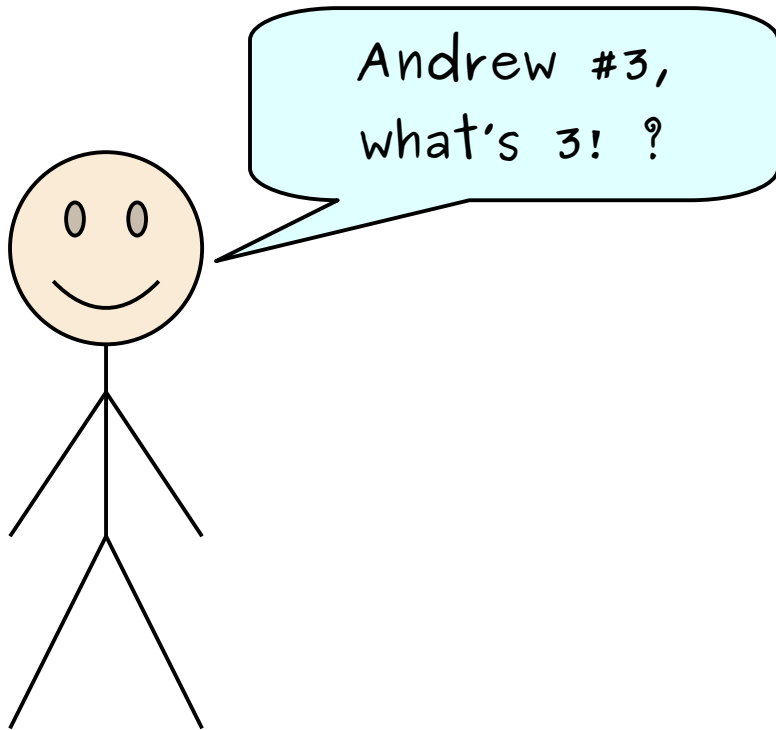


Me!

Andrews Compute Factorials

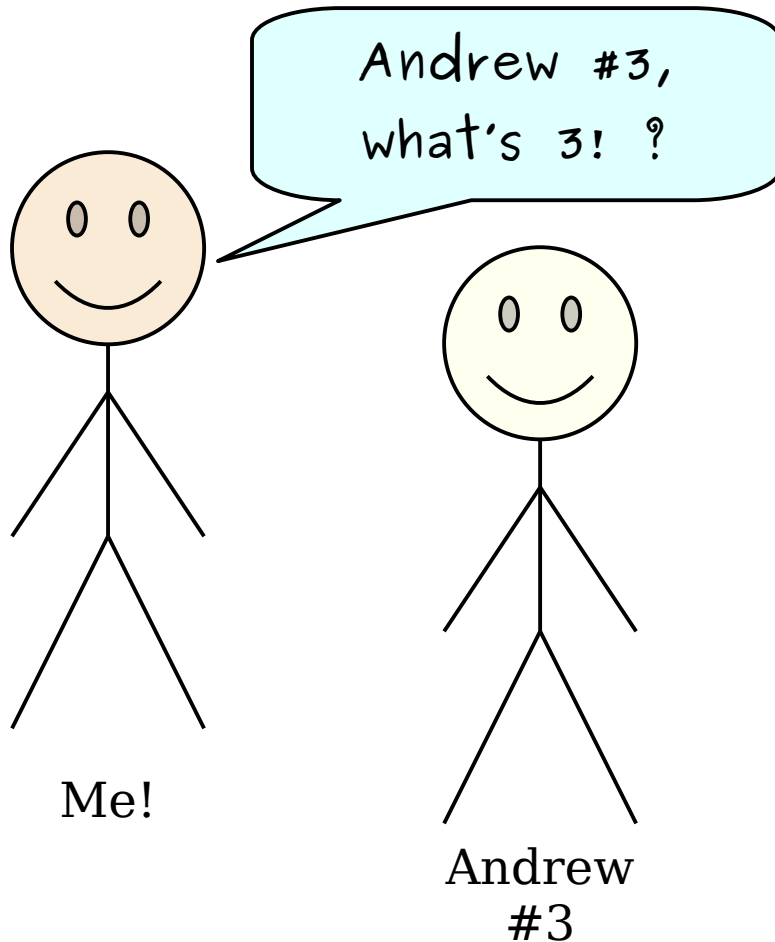


Andrews Compute Factorials

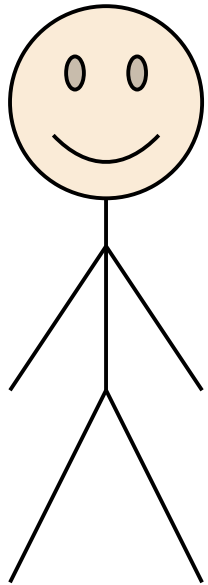


Me!

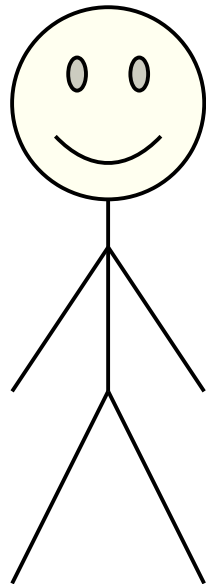
Andrews Compute Factorials



Andrews Compute Factorials



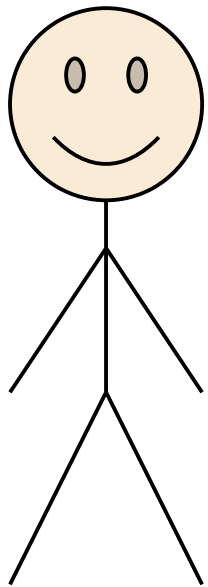
Me!



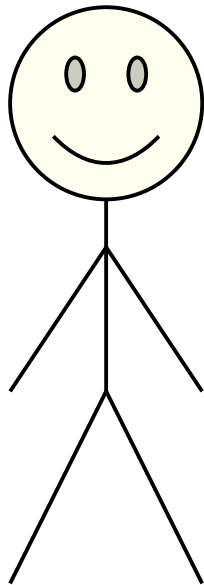
Andrew
#3

$3! = 3 \times 2!$.
I wonder what
 $2!$ is?

Andrews Compute Factorials



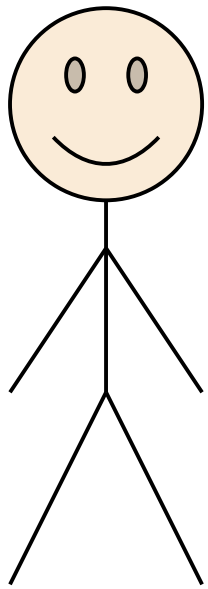
Me!



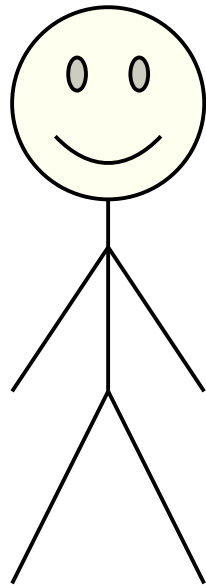
Andrew
#3

Let me ask my
friend Andrew!

Andrews Compute Factorials



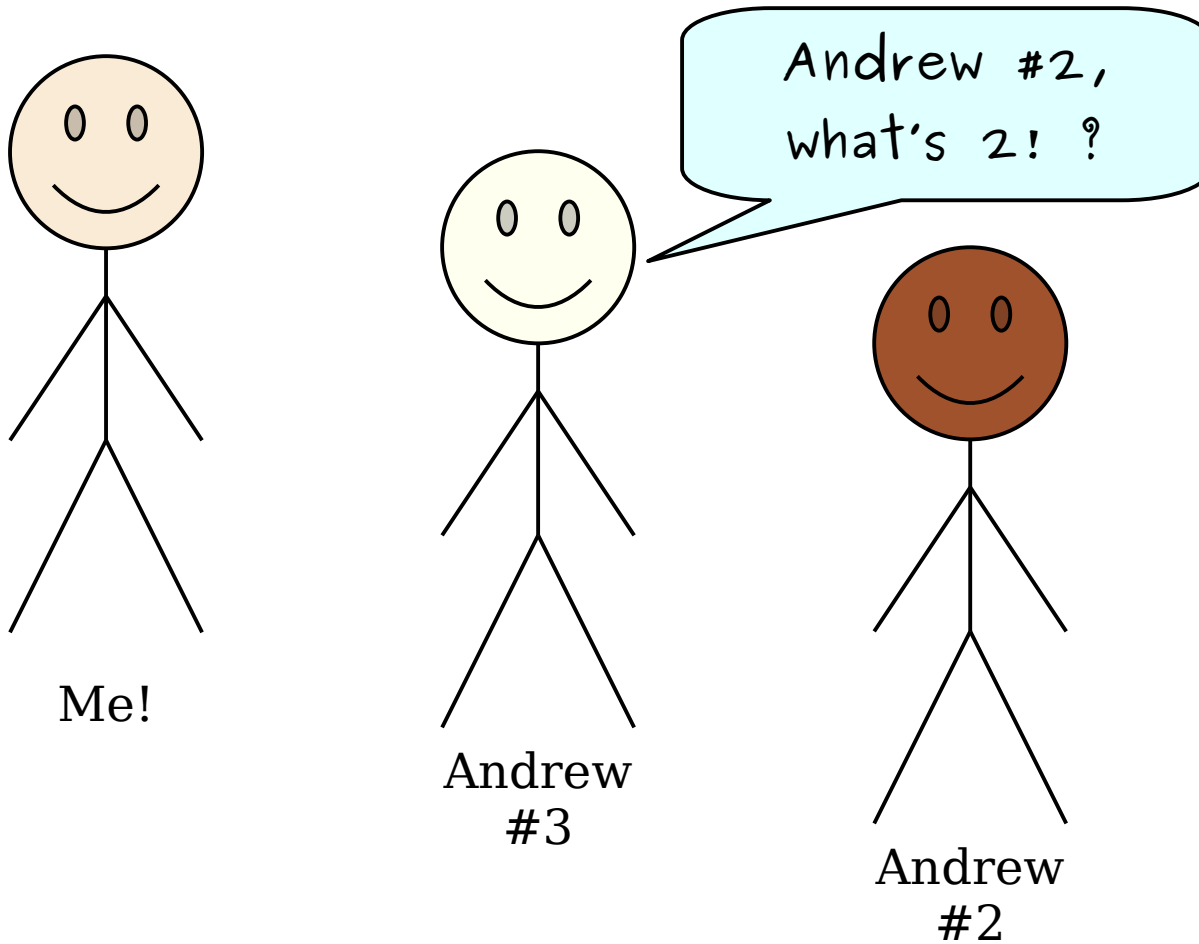
Me!



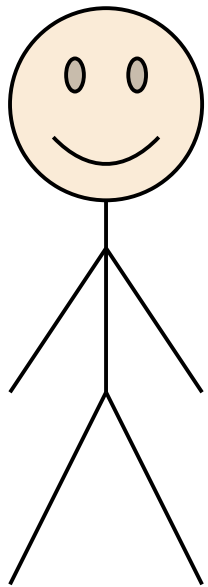
Andrew
#3

Andrew #2,
what's $2!$?

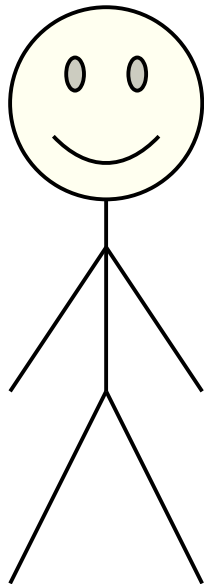
Andrews Compute Factorials



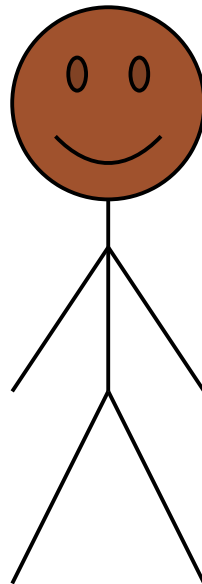
Andrews Compute Factorials



Me!



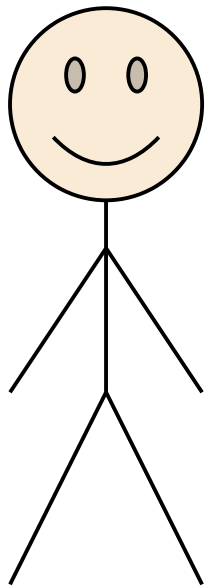
Andrew
#3



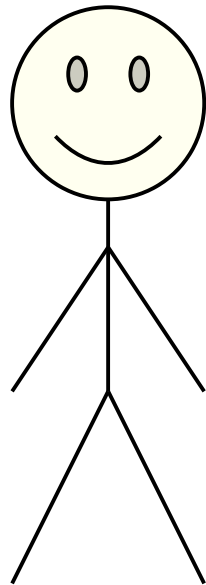
Andrew
#2

$2! = 2 \times 1!$.
I wonder what
 $1!$ is?

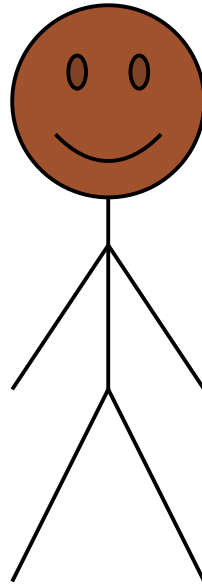
Andrews Compute Factorials



Me!



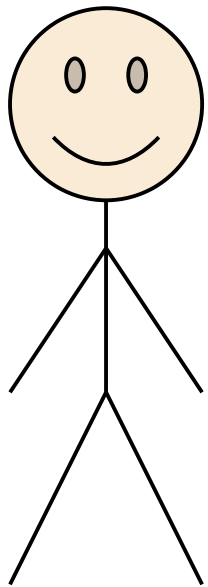
Andrew
#3



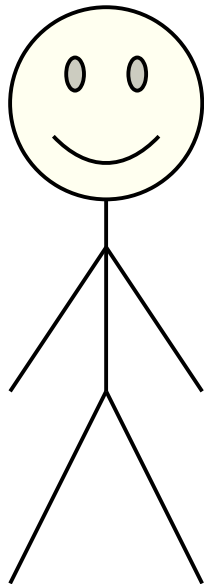
Andrew
#2

Let me ask my
friend Andrew!

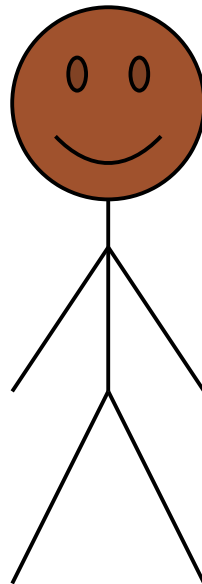
Andrews Compute Factorials



Me!



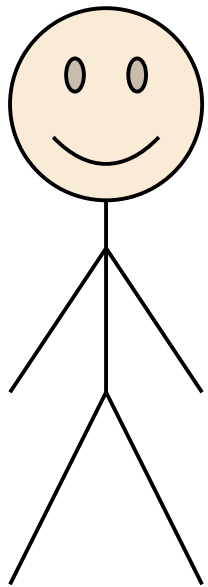
Andrew
#3



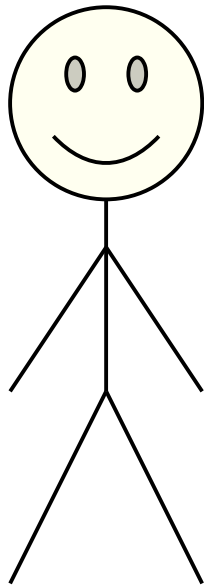
Andrew
#2

Andrew #1,
what's $1!$?

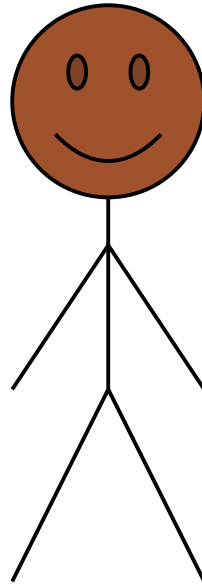
Andrews Compute Factorials



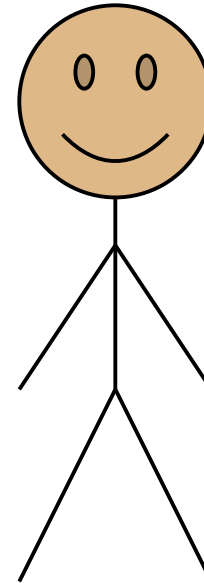
Me!



Andrew
#3

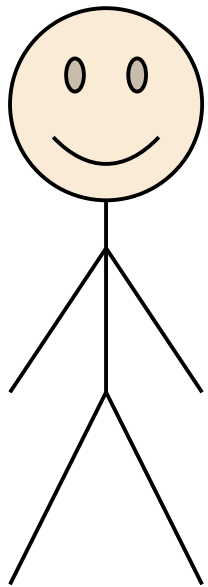


Andrew
#2

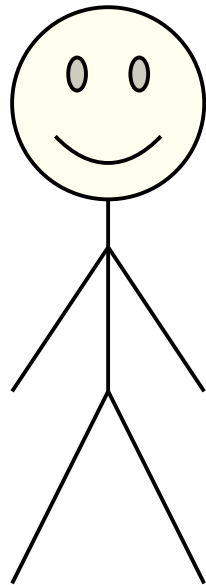


Andrew
#1

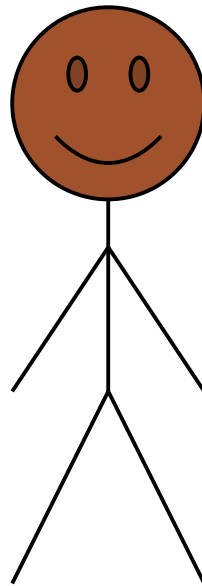
Andrews Compute Factorials



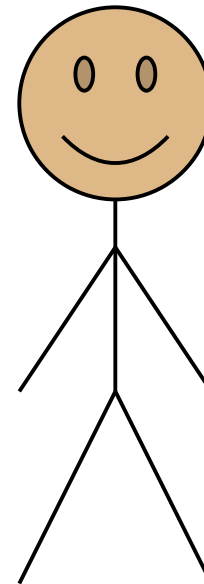
Me!



Andrew
#3



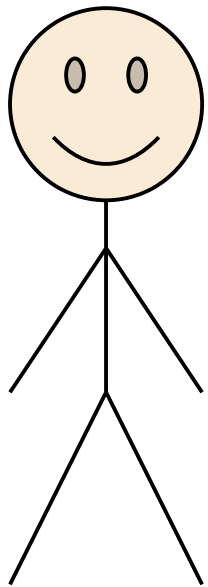
Andrew
#2



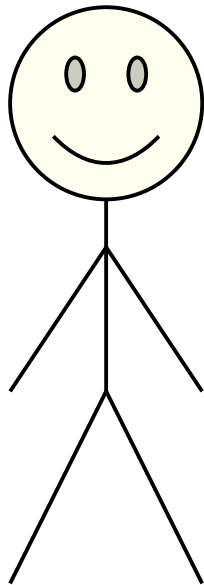
Andrew
#1

$1! = 1 \times 0!$.
I wonder
what $0!$ is?

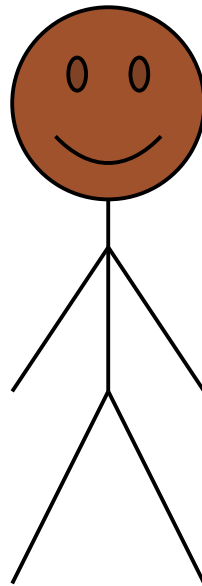
Andrews Compute Factorials



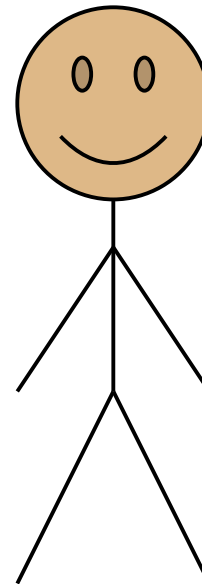
Me!



Andrew
#3



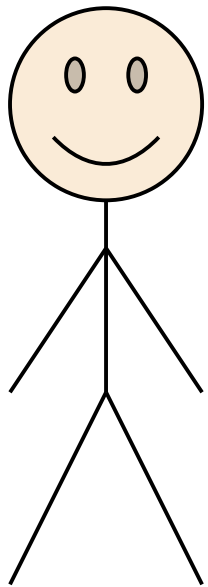
Andrew
#2



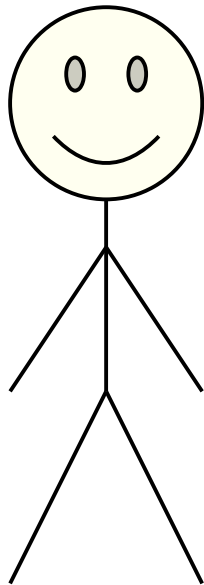
Andrew
#1

Let me ask
my friend
Andrew!

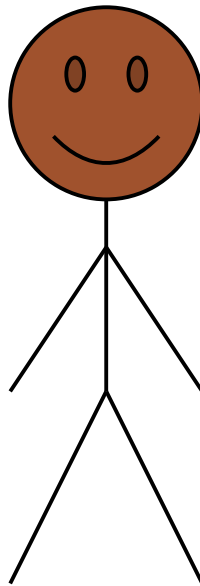
Andrews Compute Factorials



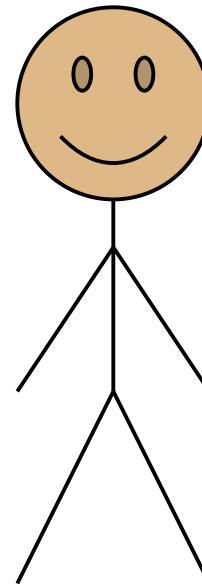
Me!



Andrew
#3



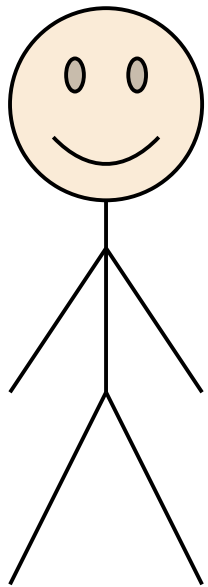
Andrew
#2



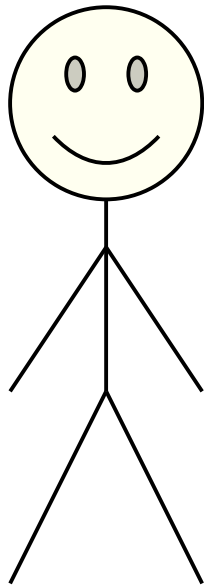
Andrew
#1

Andrew #0,
what's $0!$?

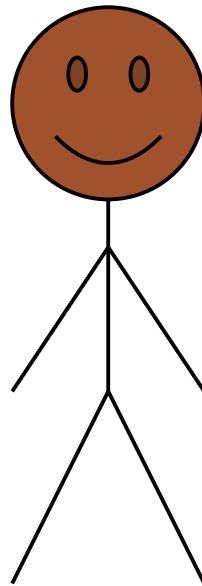
Andrews Compute Factorials



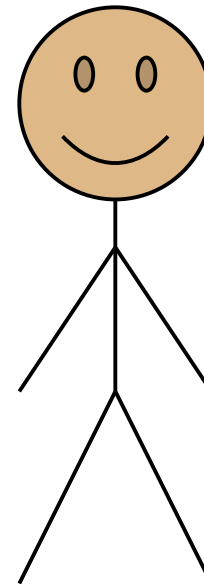
Me!



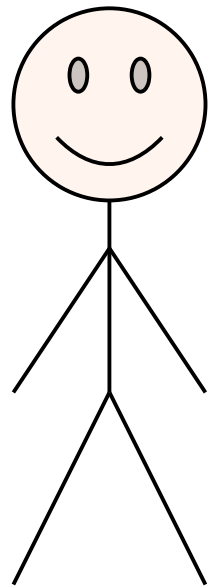
Andrew
#3



Andrew
#2

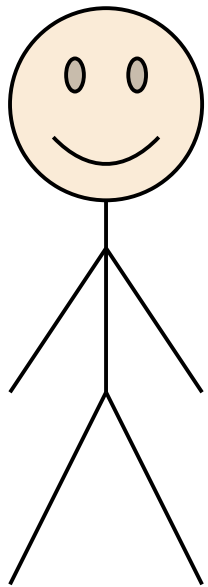


Andrew
#1

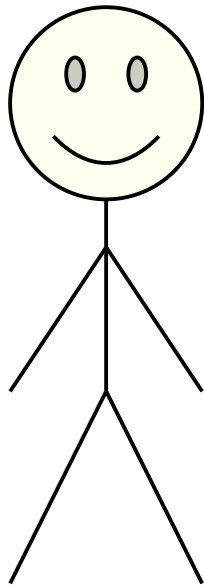


Andrew
#0

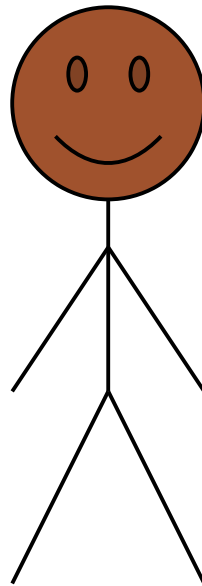
Andrews Compute Factorials



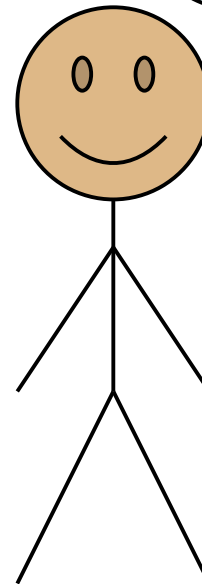
Me!



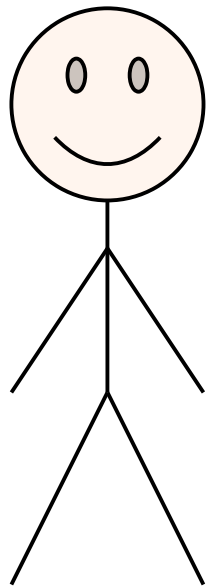
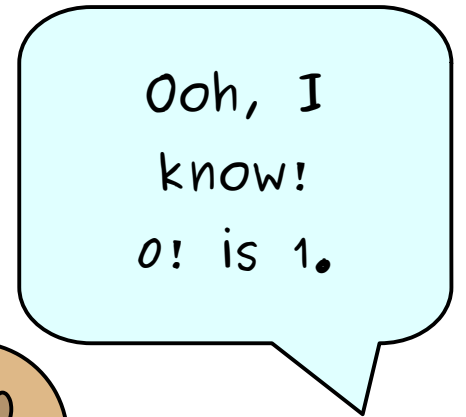
Andrew
#3



Andrew
#2

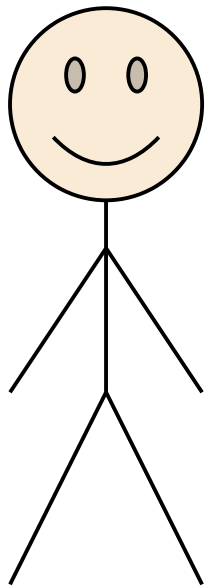


Andrew
#1

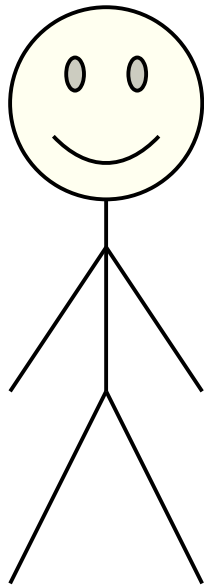


Andrew
#0

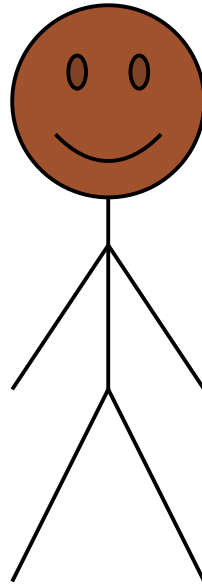
Andrews Compute Factorials



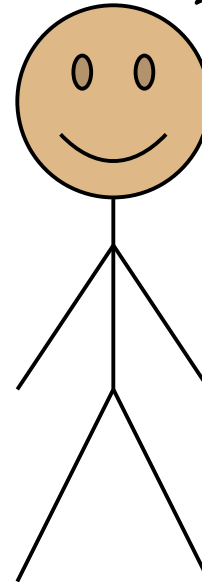
Me!



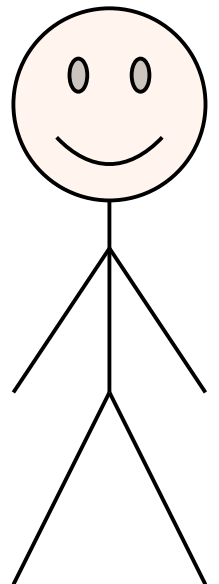
Andrew
#3



Andrew
#2

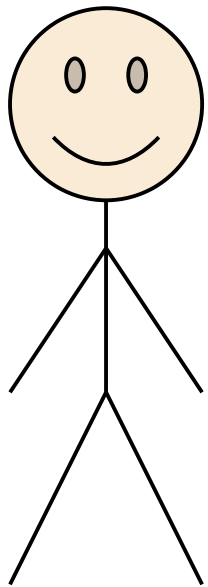


Andrew
#1

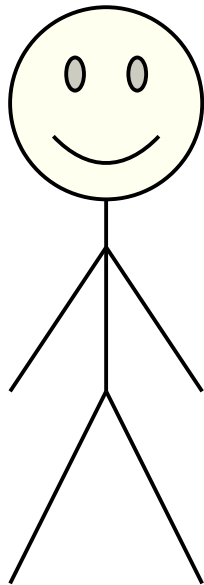


Andrew
#0

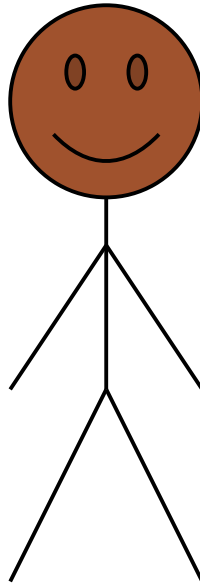
Andrews Compute Factorials



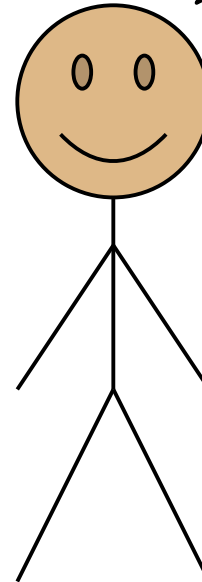
Me!



Andrew
#3



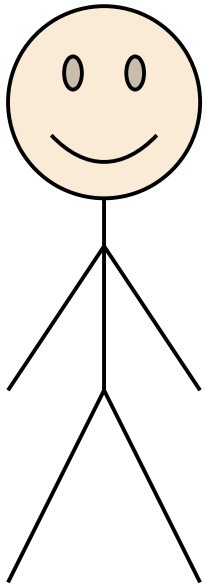
Andrew
#2



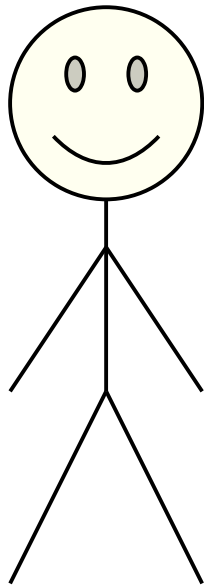
Andrew
#1

Thanks,
Andrew #0.

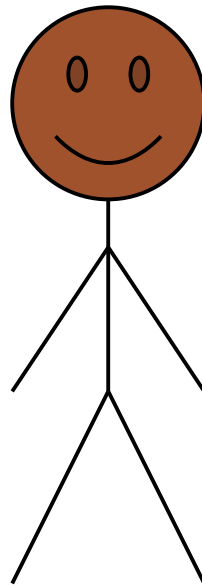
Andrews Compute Factorials



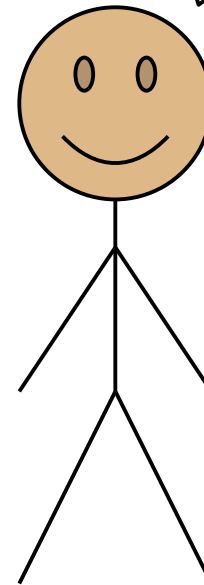
Me!



Andrew
#3



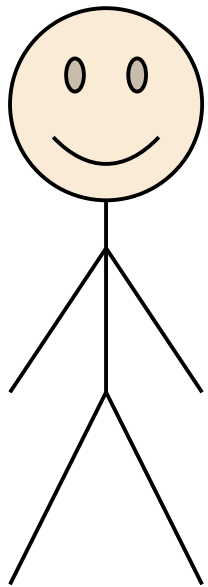
Andrew
#2



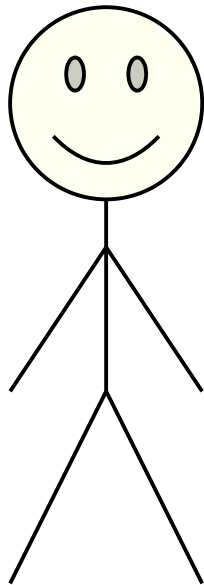
Andrew
#1

Because $0! = 1$ and
 $1! = 1 \times 0!$, the
answer is $1! = 1$.

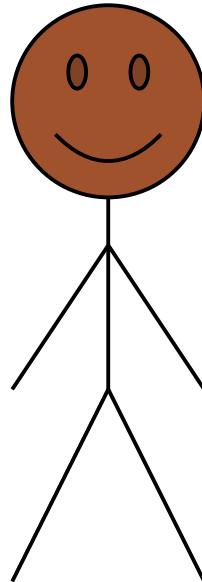
Andrews Compute Factorials



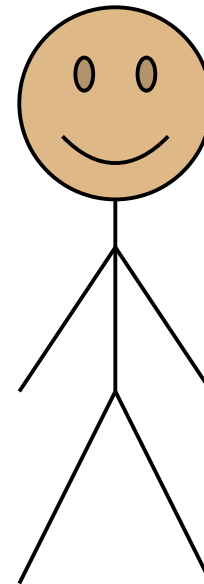
Me!



Andrew
#3

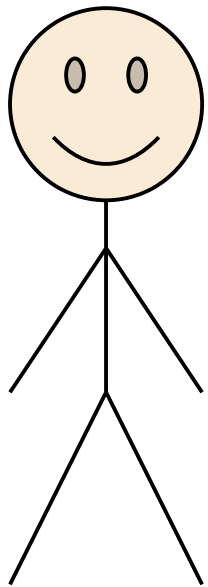


Andrew
#2

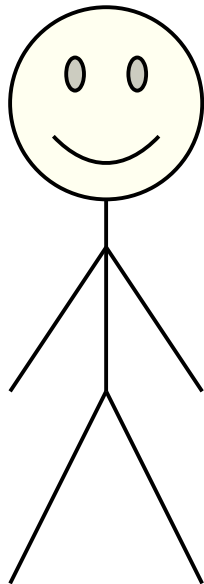


Andrew
#1

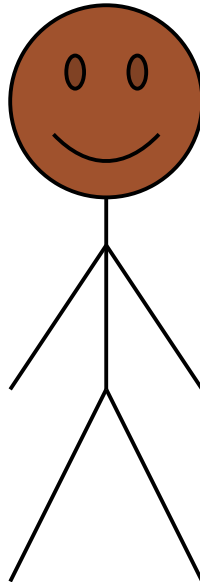
Andrews Compute Factorials



Me!



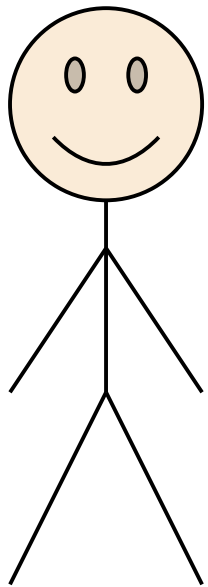
Andrew
#3



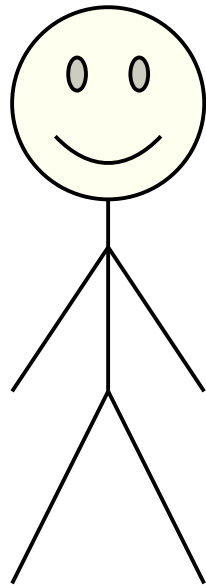
Andrew
#2



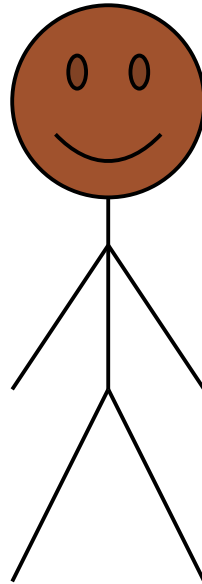
Andrews Compute Factorials



Me!



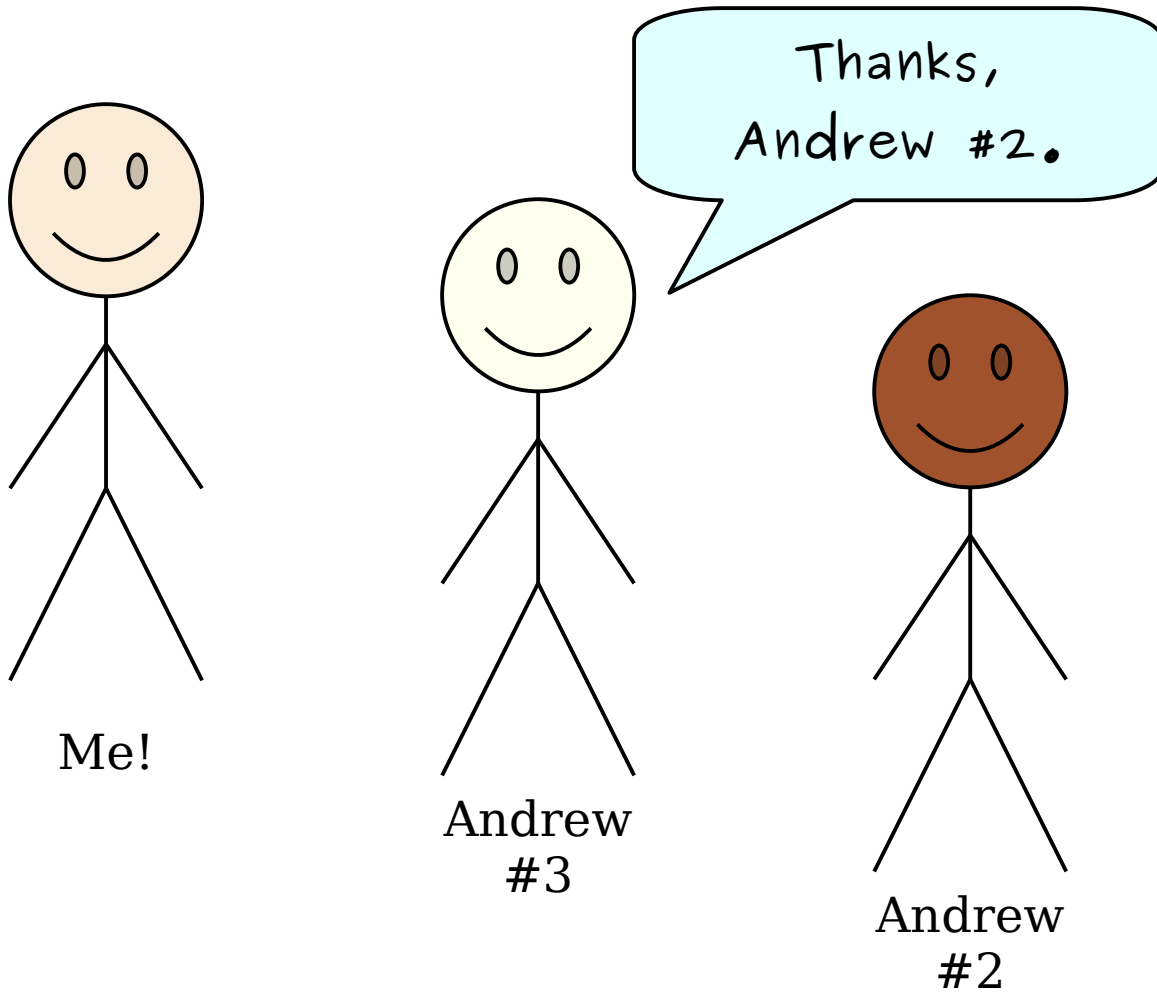
Andrew
#3



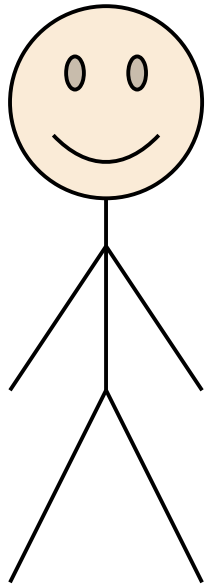
Andrew
#2

Because $1! = 1$ and
 $2! = 2 \times 1!$, the
answer is $2! = 2$.

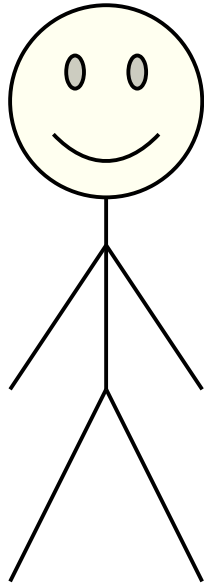
Andrews Compute Factorials



Andrews Compute Factorials



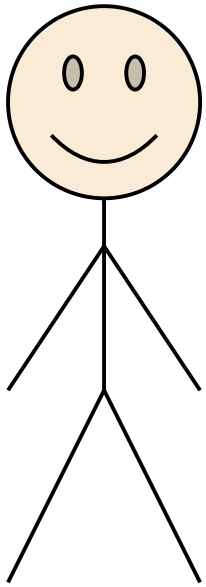
Me!



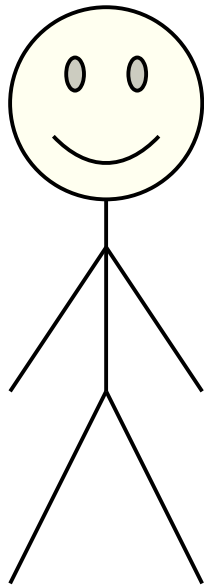
Andrew
#3



Andrews Compute Factorials



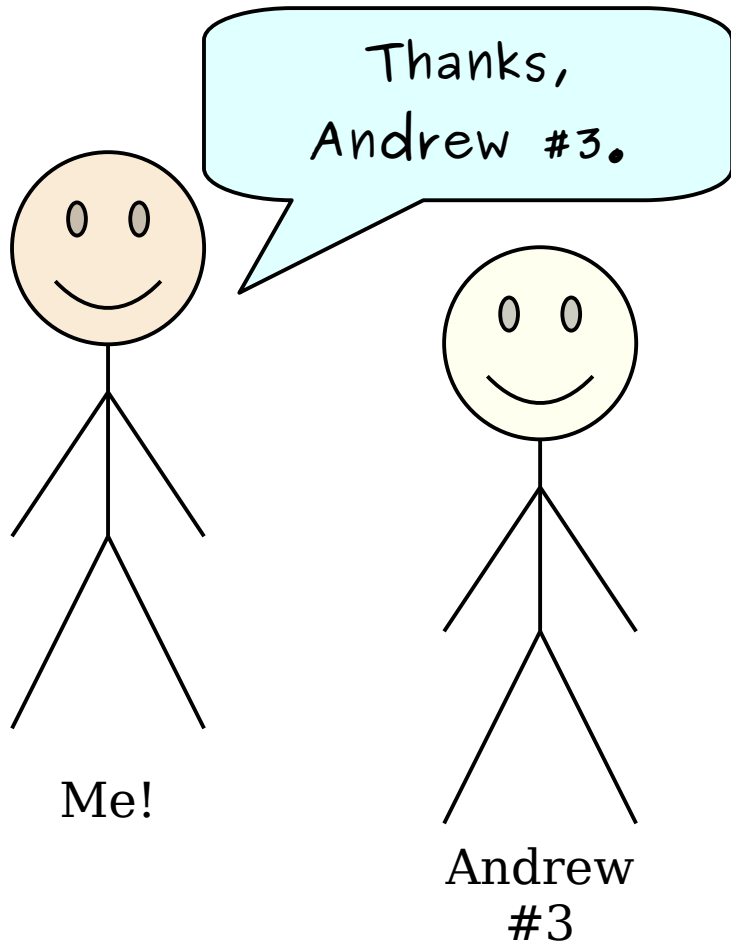
Me!



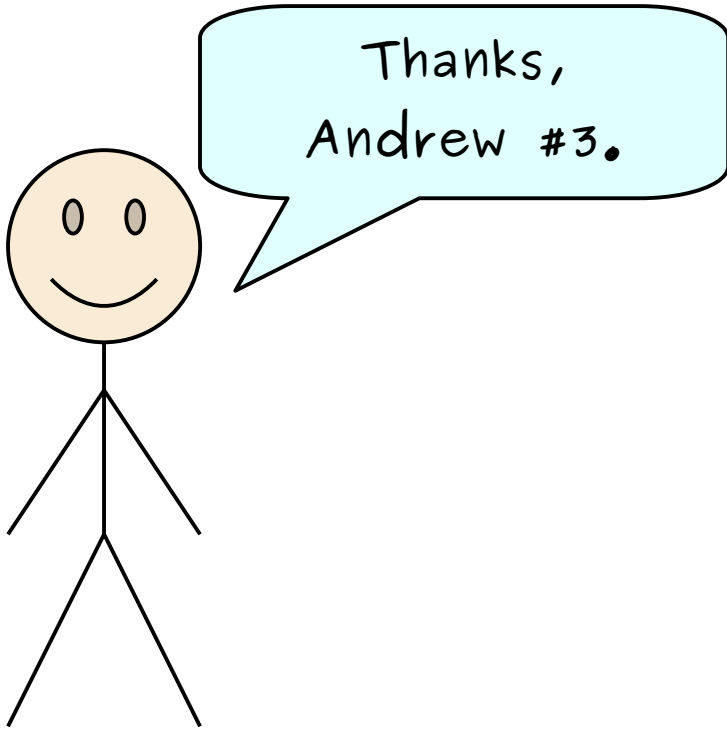
Andrew
#3

Because $2! = 2$ and
 $3! = 3 \times 2!$, the
answer is $3! = 6$.

Andrews Compute Factorials

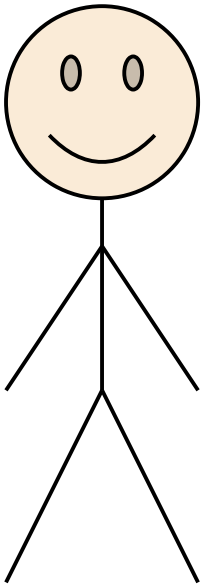


Andrews Compute Factorials



Me!

Andrews Compute Factorials



Me!

There are multiple people,
each named Andrew, but they're
not the same person.

Each Andrew is tasked with
computing a different number
factorial.

Each Andrew 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);
```

```
        }
```

```
    }
```

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

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

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

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

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

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

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

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);
```

```
        }
```

```
    }
```

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

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;  
}
```

6

int nFact

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***.

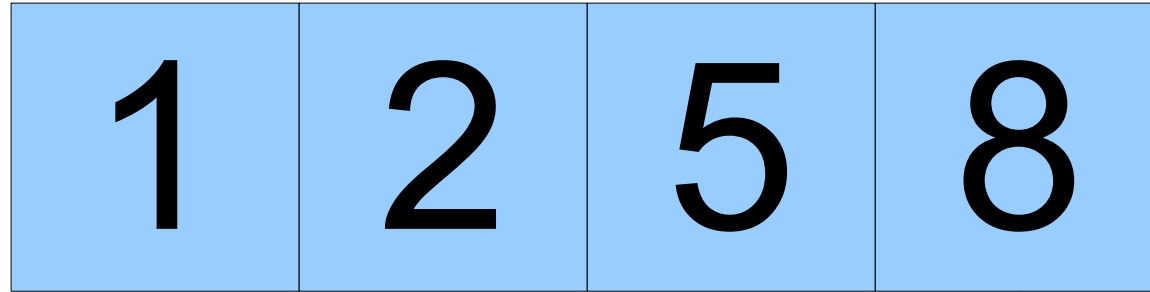
Summing Up Digits

- 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!

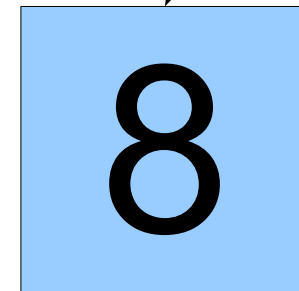
Summing Up Digits



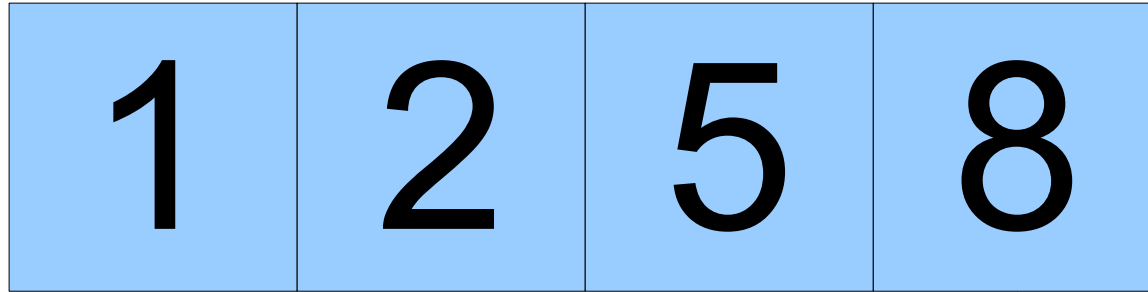
The sum of the digits of
this number is equal to...

the sum of the digits of
this number...

plus this number.



Summing Up Digits



`sumOfDigitsOf(n)`
is equal to...

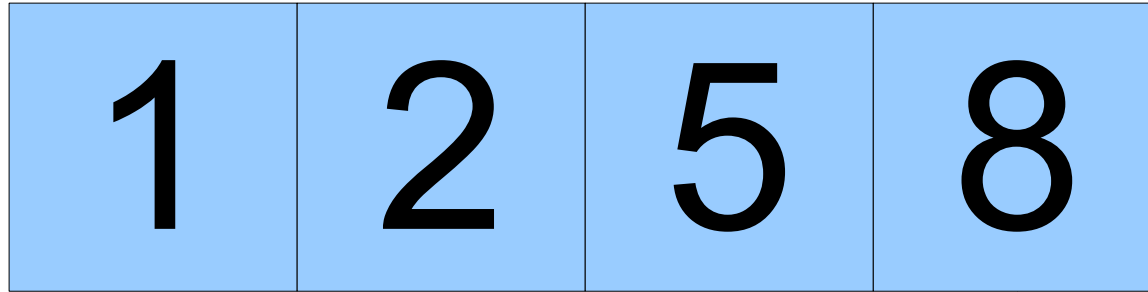
the sum of the digits of
this number...



plus this number.



Summing Up Digits



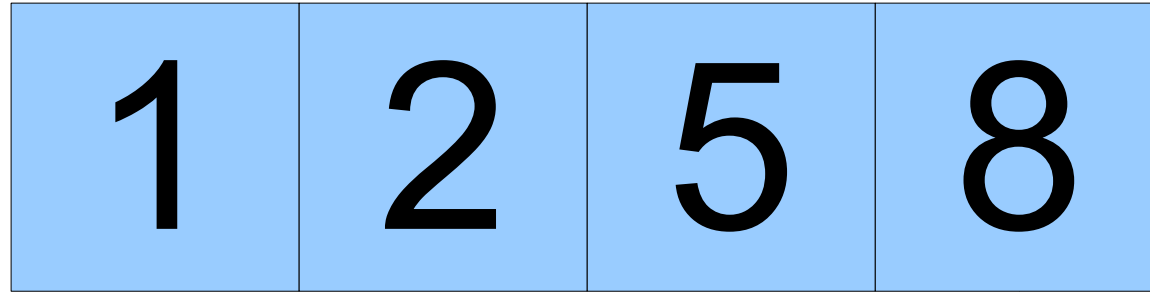
`sumOfDigitsOf(n)`
is equal to...

`sumOfDigitsOf(n / 10)`

plus this number.



Summing Up Digits

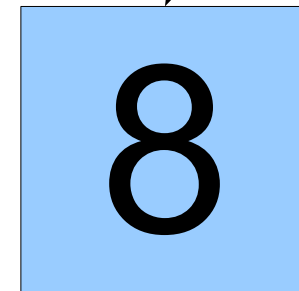


`sumOfDigitsOf(n)`
is equal to...

`sumOfDigitsOf(n / 10)`



`+ (n % 10)`



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) {
```

```
            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) {
```

```
            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) {
```

```
            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) {
```

```
                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 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) {
```

```
            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

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);
```

1

+

3

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

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!

MLK Weekend

- Some suggested reading / listening / watching recommendations:
 - “The Autobiography of Malcolm X,” as told to Alex Haley.
 - “The Ballot or the Bullet” by Malcolm X.
 - “Between the World and Me” by Ta-Nehisi Coates.
 - “The Case for Reparations” by Ta-Nehisi Coates.
 - “Debate at Cambridge Union,” James Baldwin and William F. Buckley, Jr.
 - “Do Artifacts Have Politics?” by Langdon Winner.
 - “Letter from Birmingham City Jail” by Martin Luther King, Jr.
 - “Letter from a Region in my Mind” by James Baldwin.
 - “Notes on an Imagined Plaque” by The Memory Palace.
 - “The Other America” by Martin Luther King, Jr.

Asynchronous Lecture

- We will not have class this upcoming Monday in observance of the MLK holiday.
- Monday's lecture will instead be prerecorded and available online on Canvas starting at 5PM today.
- You should watch that lecture any time before Wednesday's (in-person) lecture.

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.

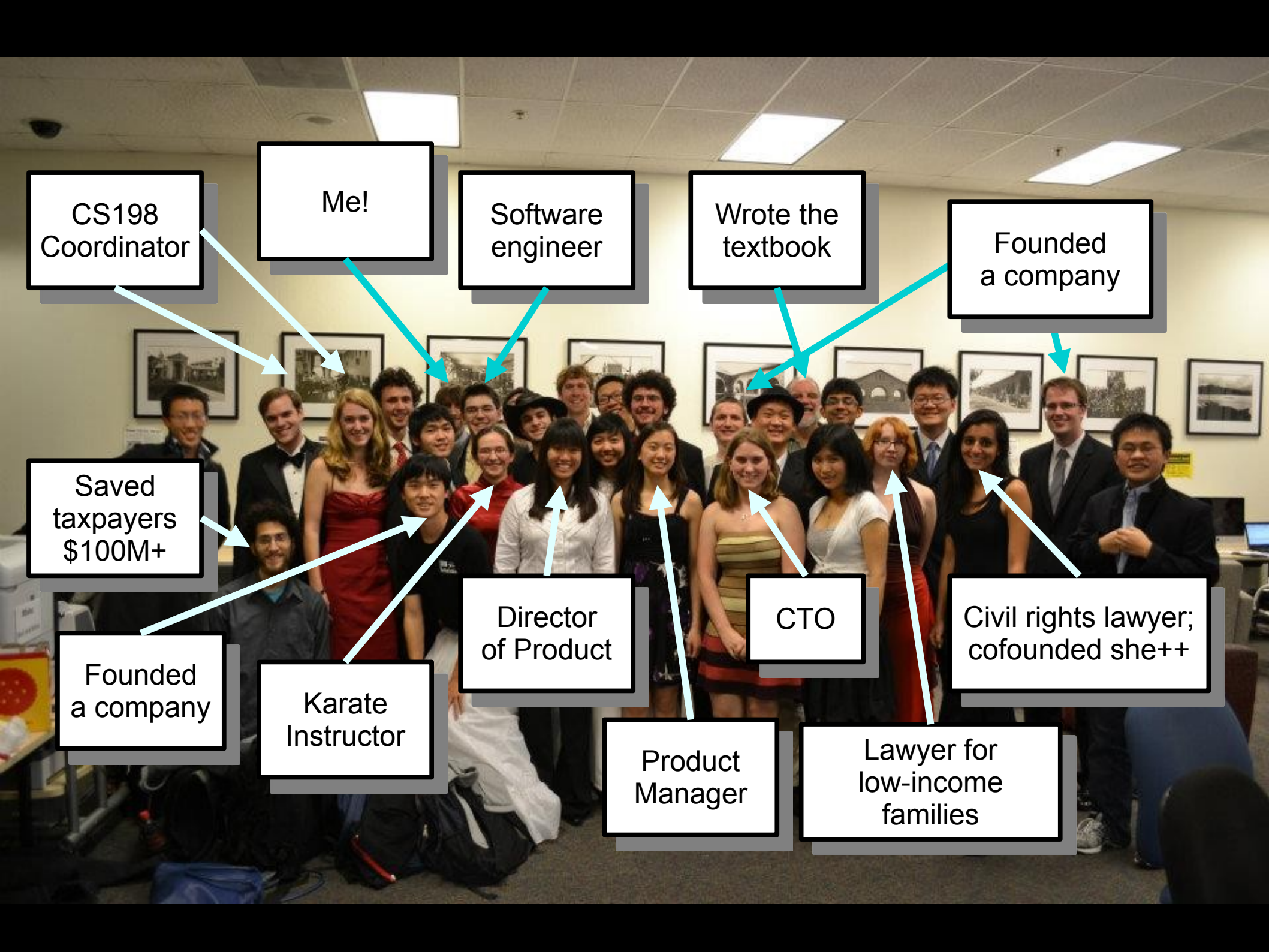
Section Signups

- Section signups are open right now. They close Sunday at 5PM.
- Sign up for section at
<https://cs198.stanford.edu/cs198/auth/default.aspx>
- Click on “CS106 Sections Login,” then choose “Section Signup.”

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 20th 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++

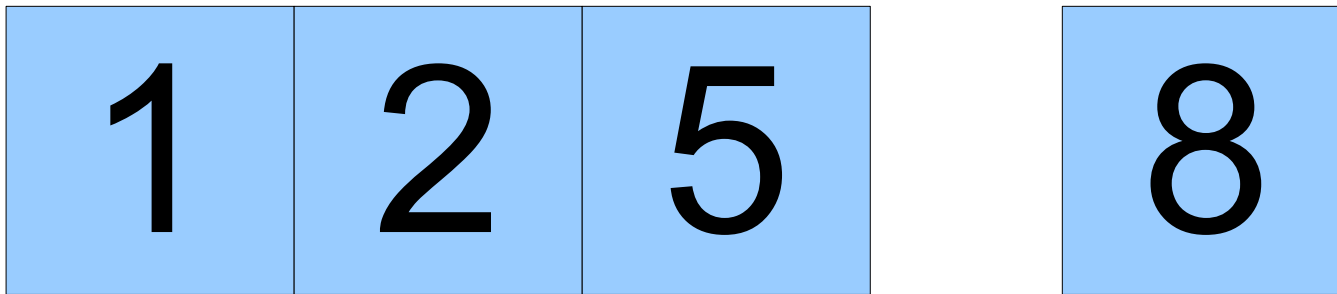
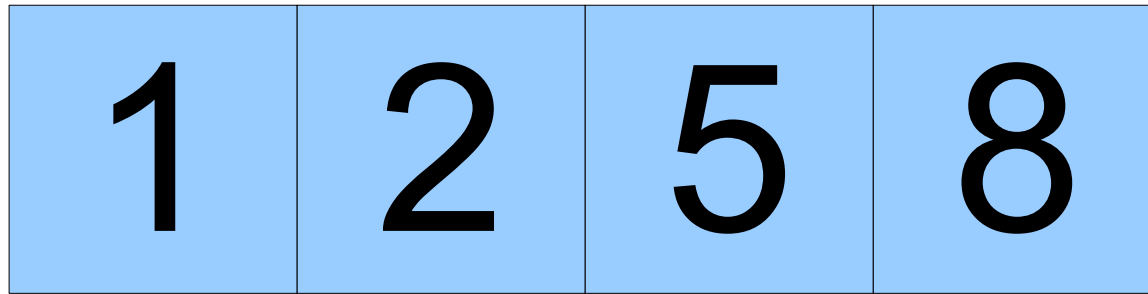
Getting Help

- ***LaIR Hours***
 - Sunday – Thursday, 7PM – 11PM Pacific.
 - Starts Monday.
 - Runs in the Durand building 3rd floor.
- ***Neel'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

str[0]

B	E	X
---	---	---

???

Answer at
<https://pollev.com/cs106bwin23>

Thinking Recursively

I	B	E	X
---	---	---	---

I

`str[0]`

B	E	X
---	---	---

`str.substr(1)`

Reversing a String

N	u	b	i	a	n		I	b	e	x
---	---	---	---	---	---	--	---	---	---	---

x	e	b	I		n	a	i	b	u	N
---	---	---	---	--	---	---	---	---	---	---



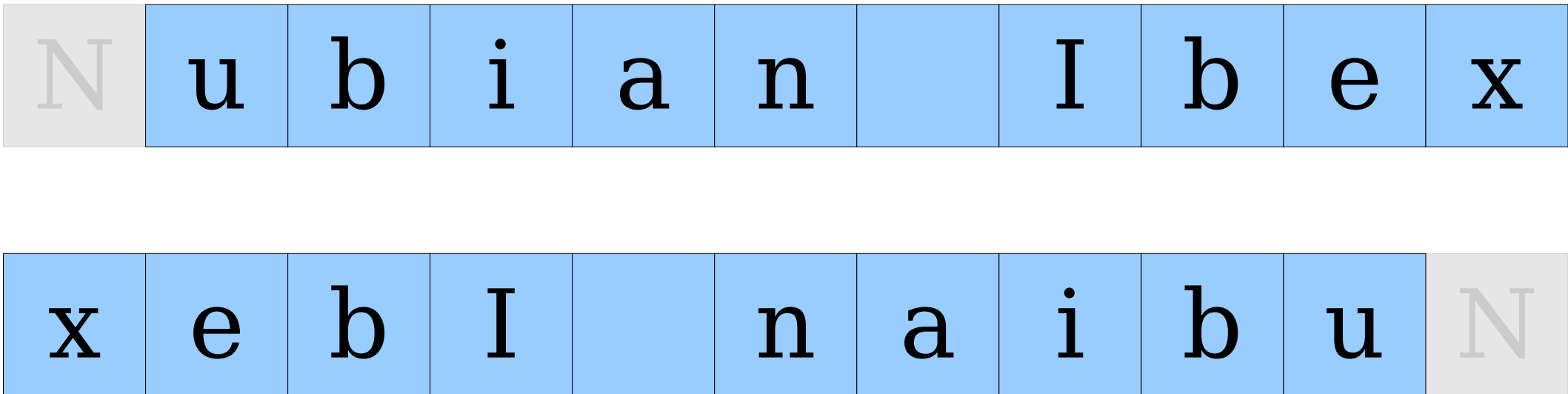
Reversing a String

N	u	b	i	a	n		I	b	e	x
x	e	b	I		n	a	i	b	u	N

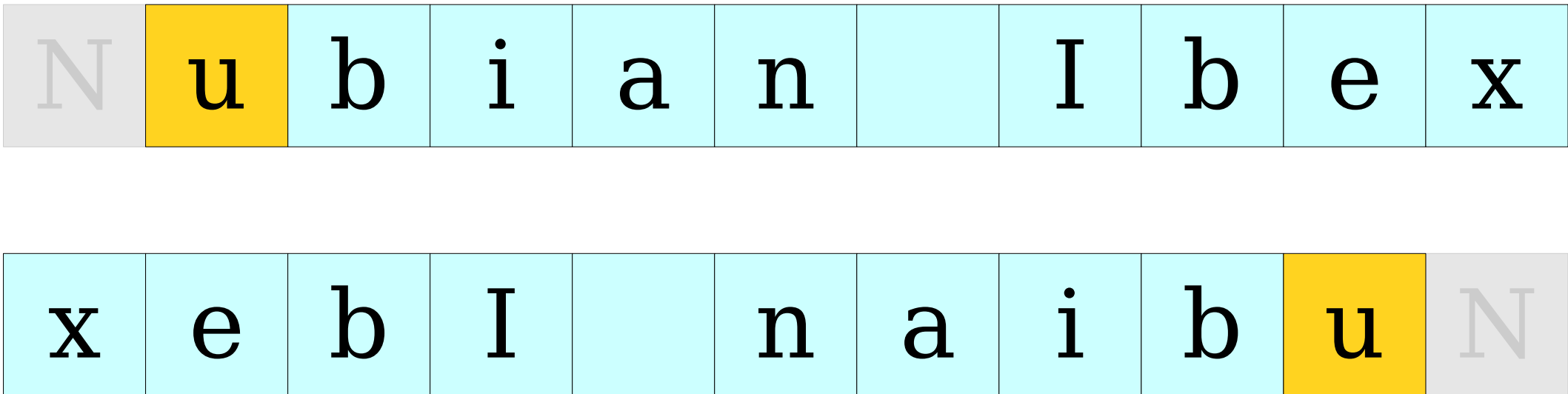
Reversing a String

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("TOP ") =`

Reversing a String Recursively

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

Reversing a String Recursively

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

`reverseOf("OP ") =`

Reversing a String Recursively

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

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

Reversing a String Recursively

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

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

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

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

`reverseOf("")` = ""

Reversing a String Recursively

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

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

`reverseOf("P ") = P`

`reverseOf("") = ""`

Reversing a String Recursively

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

`reverseOf("OP ") = PO`

`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("")` = ""

I B E X

I

`input[0]`

B E X

`input.substr(1)`

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 Axxess.
- ***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.
- ***Watch Recorded Lecture***
 - Pick a convenient time before Wednesday.