Recursion

Questions for John DeNero?
pollev.com/cs61a

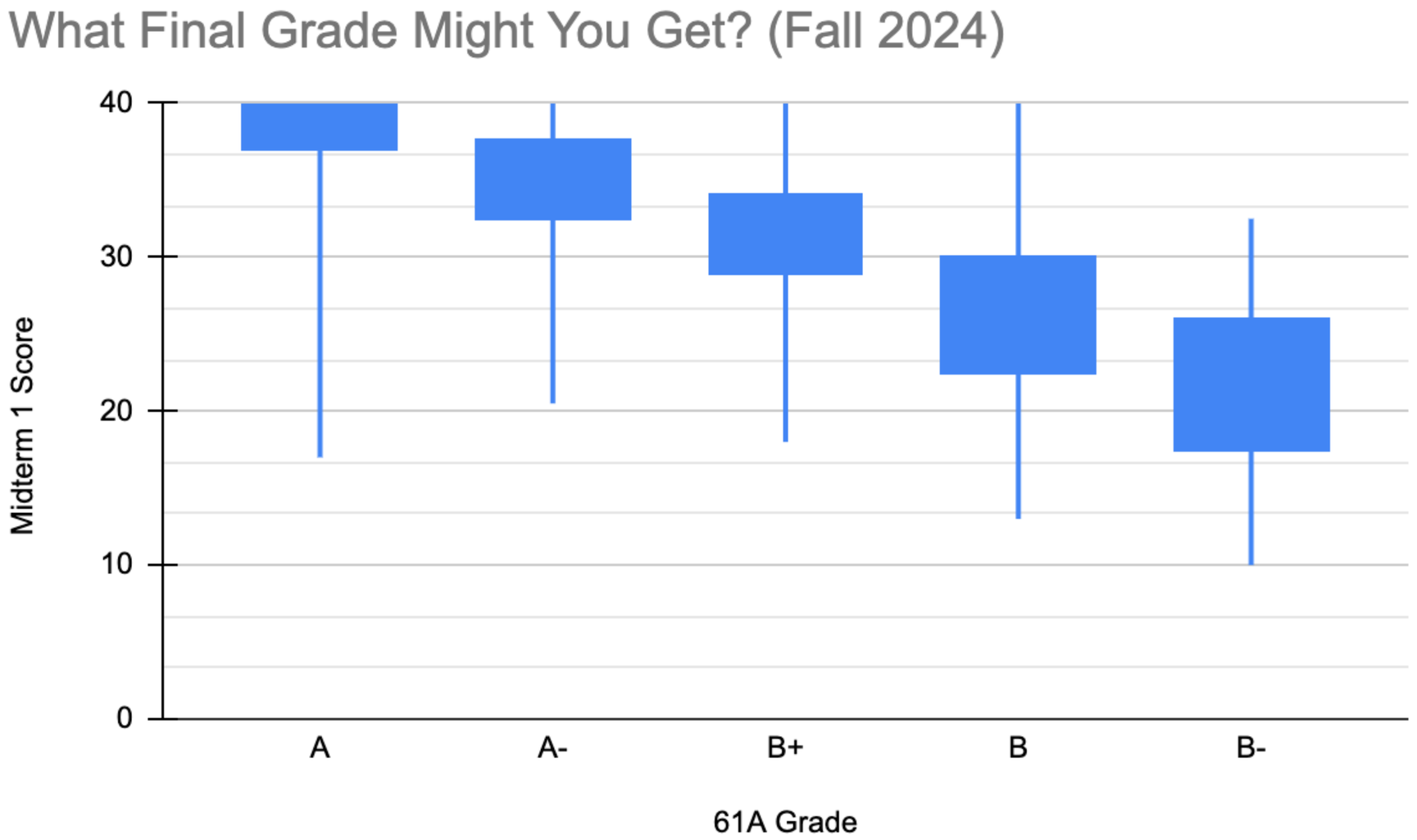
Other requests we're working on:
Oski
Jeremy Sanchez
Steve Wozniak
Instructors for future courses
(Josh Hug, Ion Stoica,
Joey Gonzalez)
Elon Musk

"Success is not final, failure is not fatal:
it is the courage to continue that counts."

– Famously Winston Churchill but actually unknown

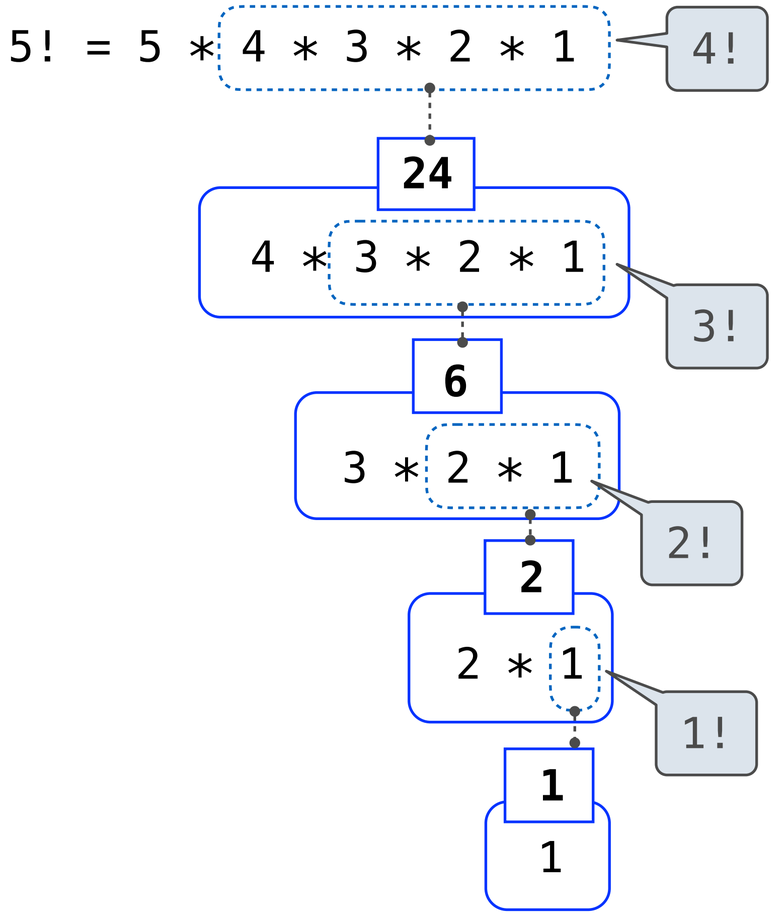
Announcements

Exam Scores


What Final Grade Might You Get? (Fall 2024)
Midterm 1 Score
40
30
20
10
0
A
A-
B+
B
B-
61A Grade

Recursive Functions

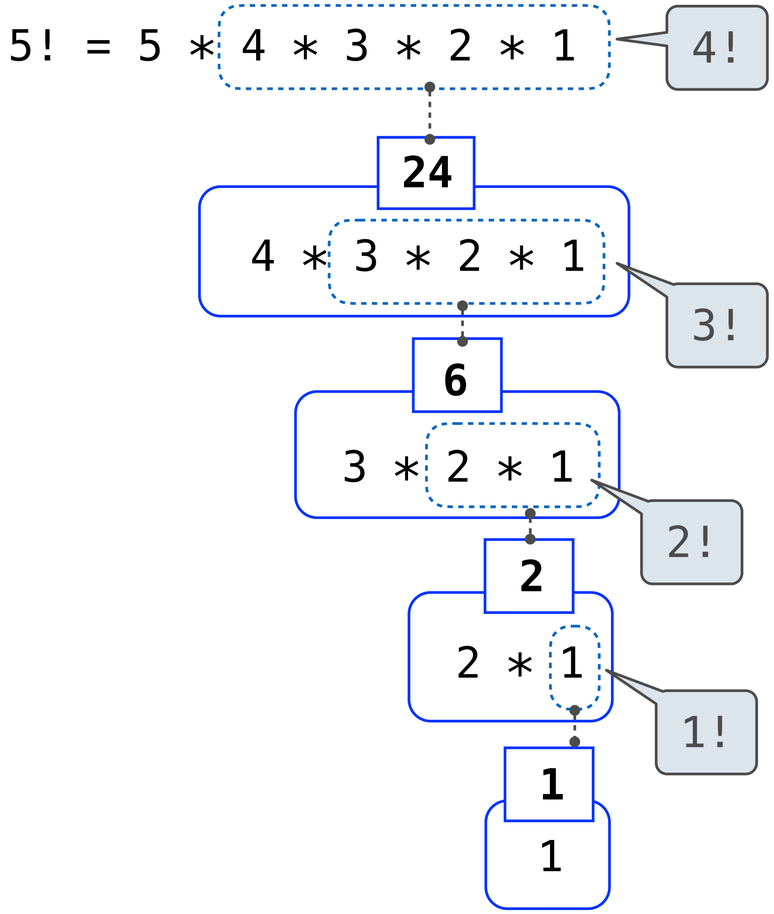
Factorial

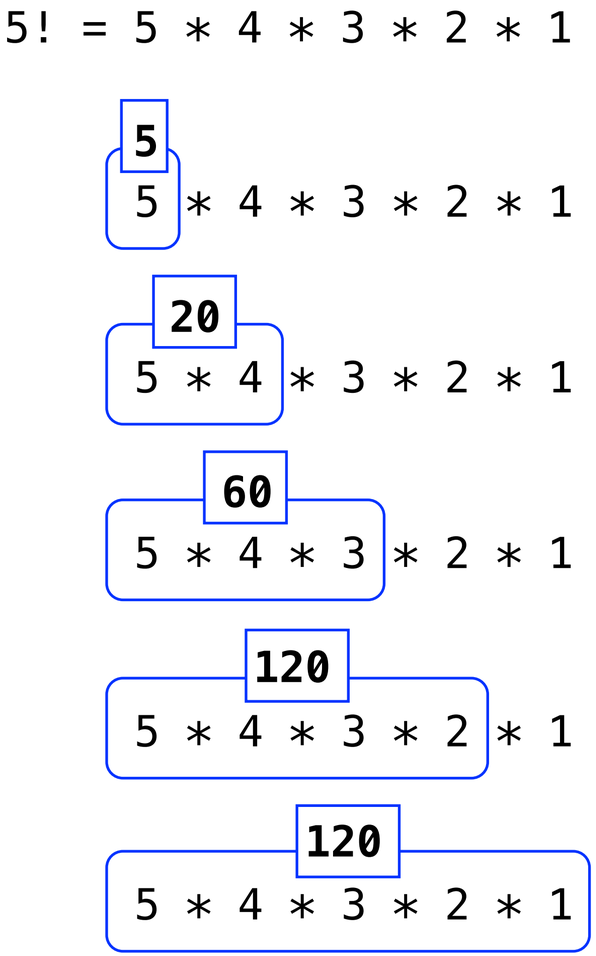



```
def fact(n):
    """Compute n factorial.

    >>> fact(5)
    120
    >>> fact(0)
    1
    """
    if n == 0 or n == 1:
        return 1
    else:
        return fact(n-1) * n
```

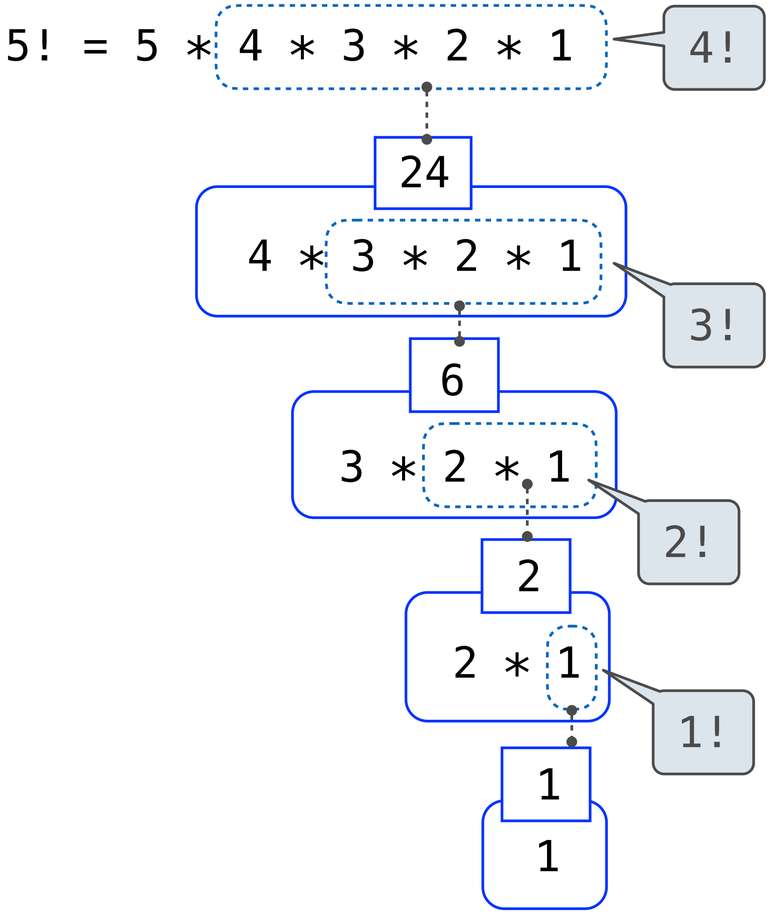
Factorial


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

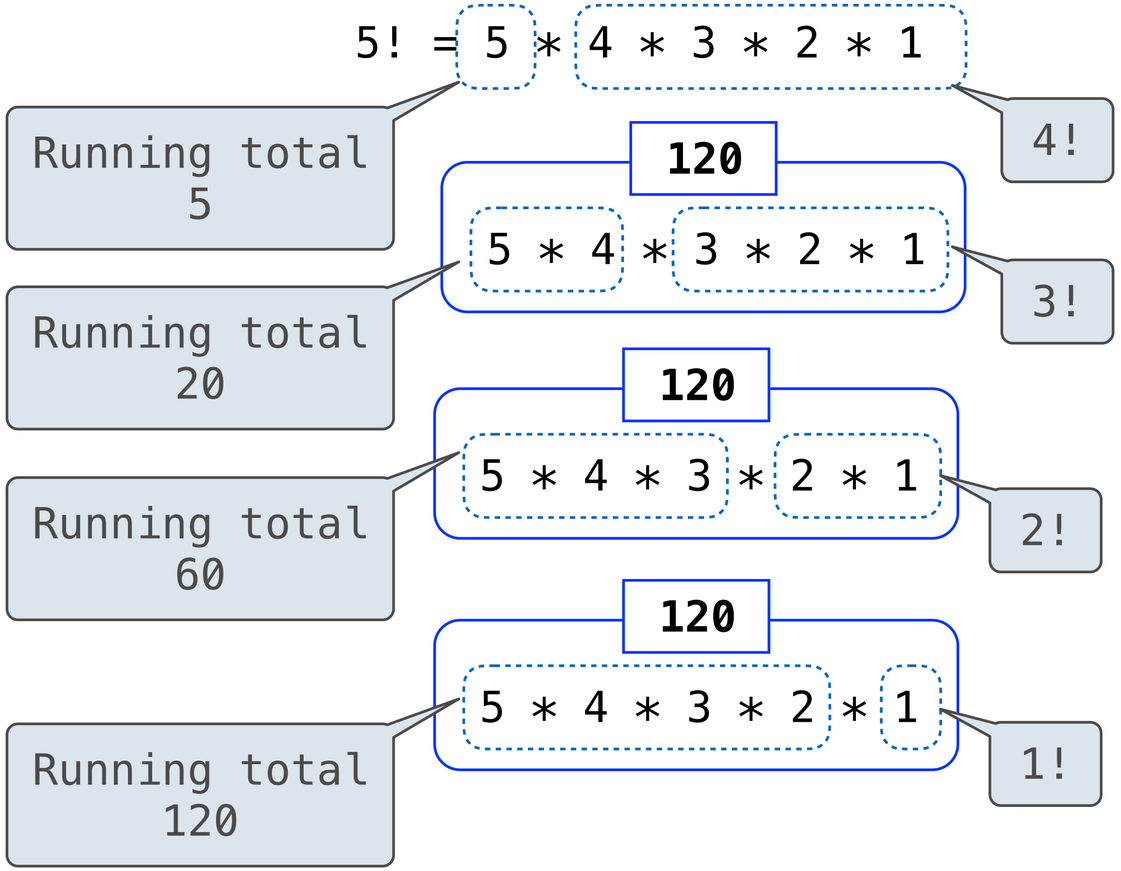

5! = 5 * 4 * 3 * 2 * 1
5
5 * 4 * 3 * 2 * 1
20
5 * 4 * 3 * 2 * 1
60
5 * 4 * 3 * 2 * 1
120
5 * 4 * 3 * 2 * 1
120
5 * 4 * 3 * 2 * 1

Factorial

Recursive approach, version 1:



while loop, fact_tail:



Call Expression Evaluation Procedure: Recursion!

Expression: describes how to compute something, evaluates to a **value**

Call Expression

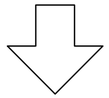
```
mul(add(4, mul(4, 6)), add(3, 5))
```

Operator *Operand* *Operand*

Operator and Operands are also expressions!

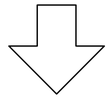
Evaluation Procedure for call expressions

(1) Evaluate operator



function

(2) Evaluate each operand



argument

(3) **Apply** the **function** to the **arguments**

Converting Iteration to Recursion

Discussion Question: Play Twenty-One

Rewrite play as a recursive function without a while statement.

- Do you need to define a new inner function? Why or why not? If so, what are its arguments?
- What is the base case and what is returned for the base case?

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    n = 0
    who = 0  # Player 0 goes first
    while n < goal:
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
    return who
```

```
def play(strategy0, strategy1, goal=21):
    """Play twenty-one and return the winner.

    >>> play(two_strat, two_strat)
    1
    """
    def f(n, who):
        if n >= goal:
            return who
        if who == 0:
            n = n + strategy0(n)
            who = 1
        elif who == 1:
            n = n + strategy1(n)
            who = 0
        return f(n, who)
    return f(0, 0)
```

Twenty-One Rules

Two players alternate turns, on which they can add 1, 2, or 3 to the current total

The total starts at 0

The game end whenever the total is 21 or more

The last player to add to the total loses

Fibonacci

```
fib(n):   0, 1, 1, 2, 3, 5, 8, 13, 21,      ... ,   9,227,465

   n:   0, 1, 2, 3, 4, 5, 6,  7,  8,       ... ,          35
```

```
fib(9) =  fib(8) + fib(7)
```

How would you re-write this in terms of the other fibs we already know?

pollev.com/cs61a

```
fib(n) = _____________
```

pollev.com/cs61a

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Tree Recursion

Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call

n:	0,	1,	2,	3,	4,	5,	6,	7,	8,	... ,	35
fib(n):	0,	1,	1,	2,	3,	5,	8,	13,	21,	... ,	9,227,465

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg