Recursion

Announcements

9% of students answered Midterm 1 Problem 2 correctly

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

– Famously Winston Churchill (but actually unknown)

Code & Environments

You can refer to local names, enclosing function names, and global names

For any **expression**, you know the frames it will be **evaluated in**
without drawing the whole environment diagram

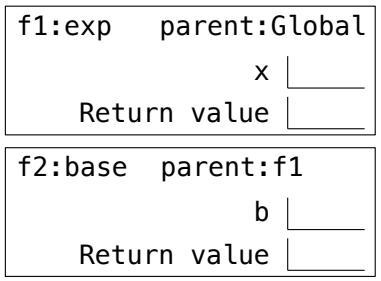
- For any expression that appears outside a **def** statement or **lambda** expression, the environment is just the global frame
- For other expressions: the environment has a frame for every enclosing **def** statement or **lambda** expression, ordered from inner to outer

```
def exp(x):
    def base(b):
        return pow(b, x)
    return base
```

x will be found in an **exp** frame: the argument to an **exp** call

Will always be evaluated in an environment with 3 frames: **base**, **exp**, and then **global**

b will be found in a **base** frame



For any **name**, you know the frame it will be **found in**
without drawing the whole environment diagram

New Print (Fall 2026 Midterm 1)

```
def new_print(print):
    def f(x):
        value = print(x)
        if value != None:
            return print('What?')
        return x
    return f

og = print
print = new_print(print)
print = new_print(print)
og('print returned', print(2))
```

print will be found in a **new_print** frame

Eval'd in some **f, new_print, Global** env

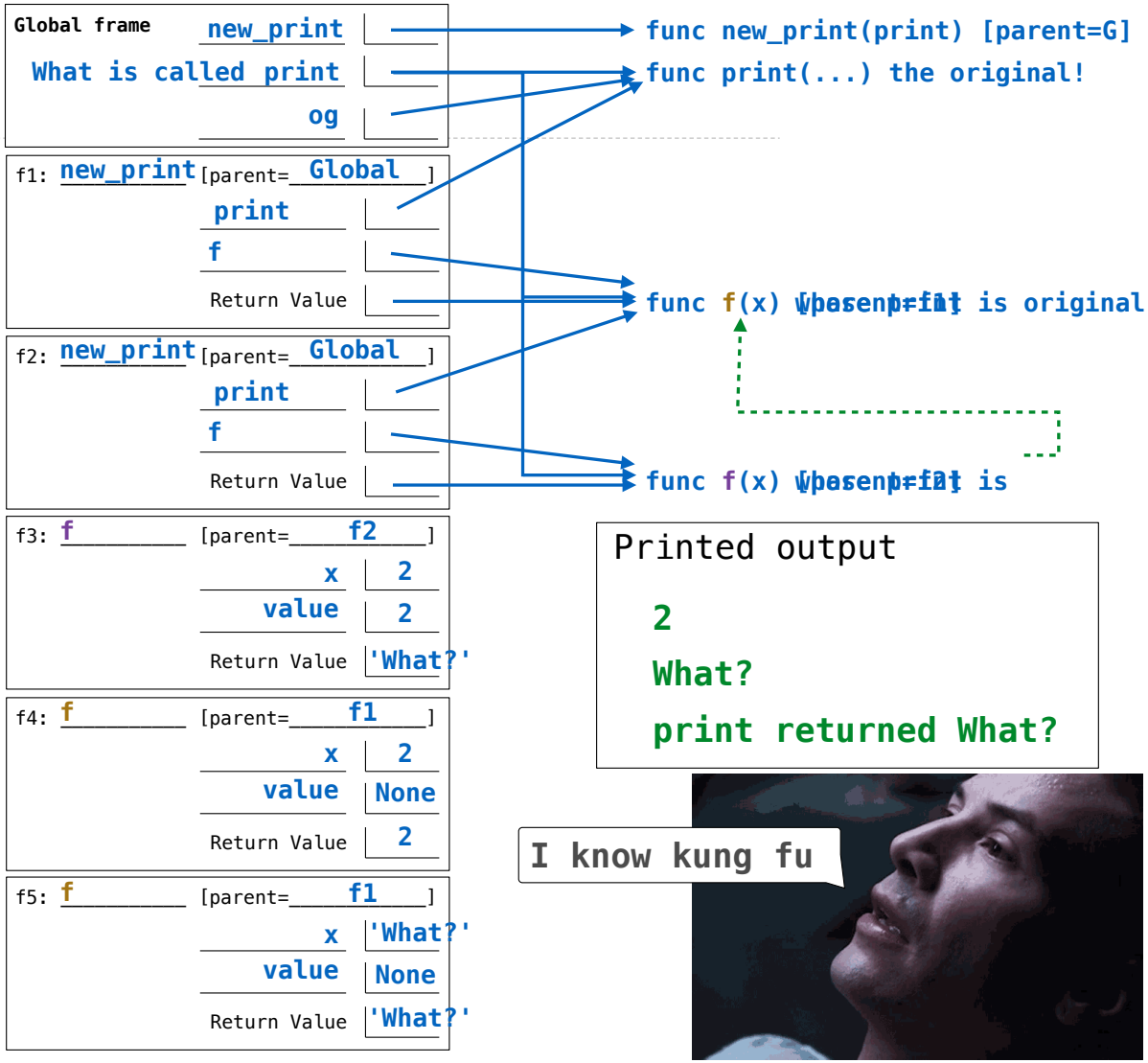
also here!

Evaluated in **global**

original **print**

f whose **print** is the original **print**

f whose **print** is (**f** whose **print** is the original **print**)

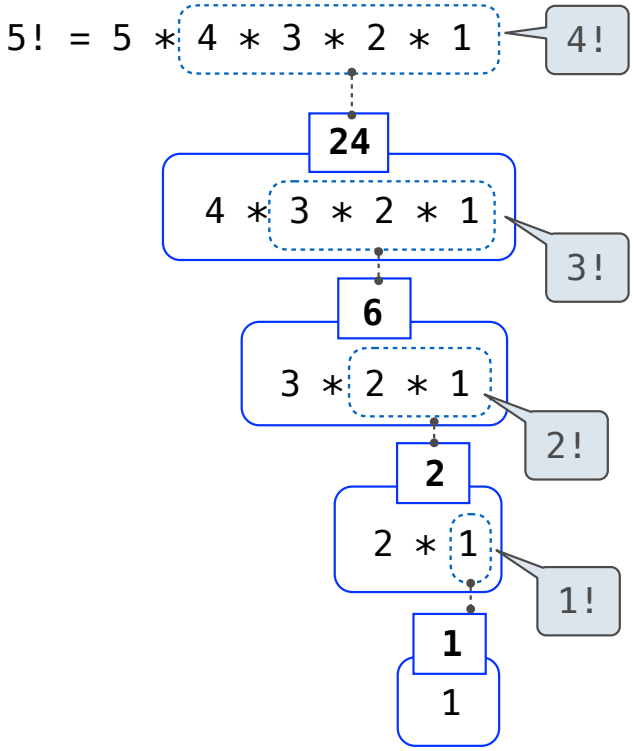


Printed output

```
2
What?
print returned What?
```

Recursive Functions

Factorial

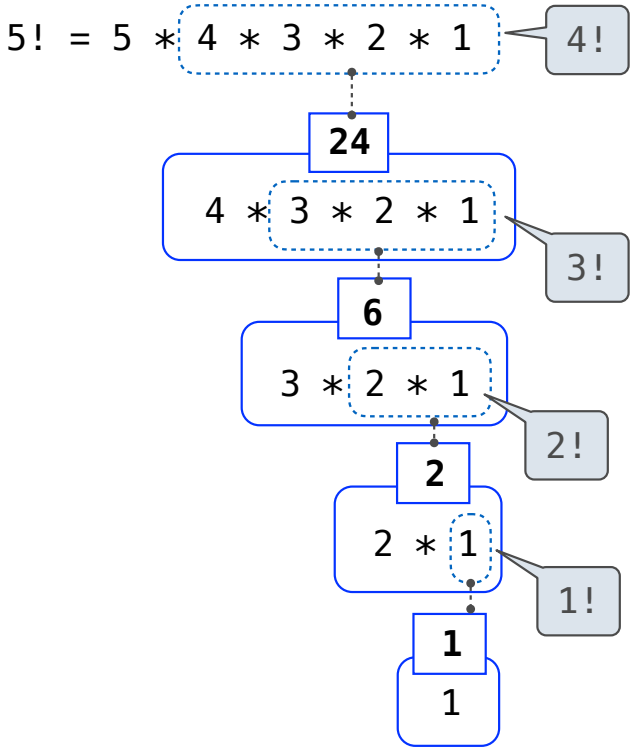


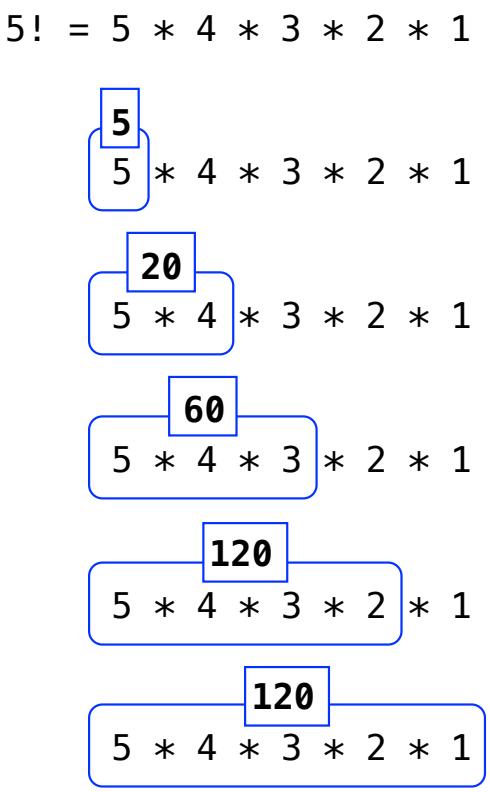

```
def fact(n):
    """Compute n factorial.

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


https://pythontutor.com/cp/composingprograms.html#code=def%20fact%28n%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%20or%20n%20%3D%3D%201%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20fact%28n-1%29%20*%20n%0A%0Afact%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

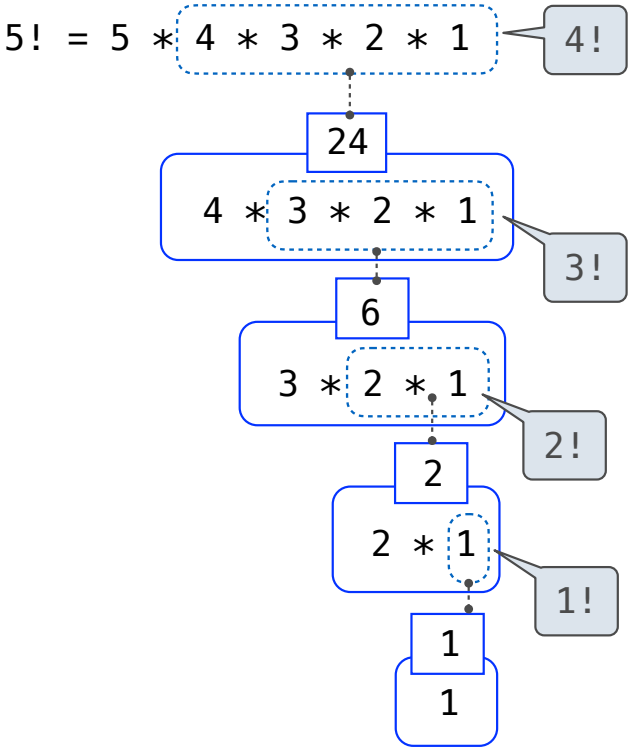
Factorial


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

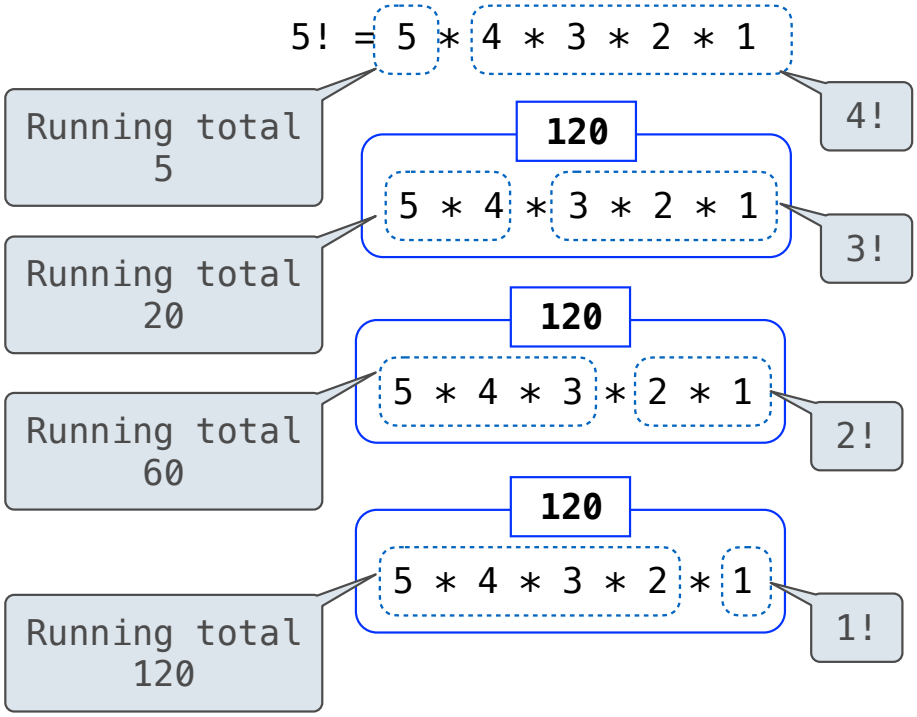

5! = 5 * 4 * 3 * 2 * 1
5
5 * 4 * 3 * 2 * 1
20
5 * 4 * 3 * 2 * 1
60
5 * 4 * 3 * 2 * 1
120
5 * 4 * 3 * 2 * 1
120
5 * 4 * 3 * 2 * 1

Factorial

Recursive approach, version 1:



while loop, fact_tail:



Converting Iteration to Recursion

Idea: The state of the while loop is passed as arguments

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

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(some_strat, some_other_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(some_strat, some_other_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)
```