

Recursion

Announcements

Self-Reference

Returning a Function Using Its Own Name

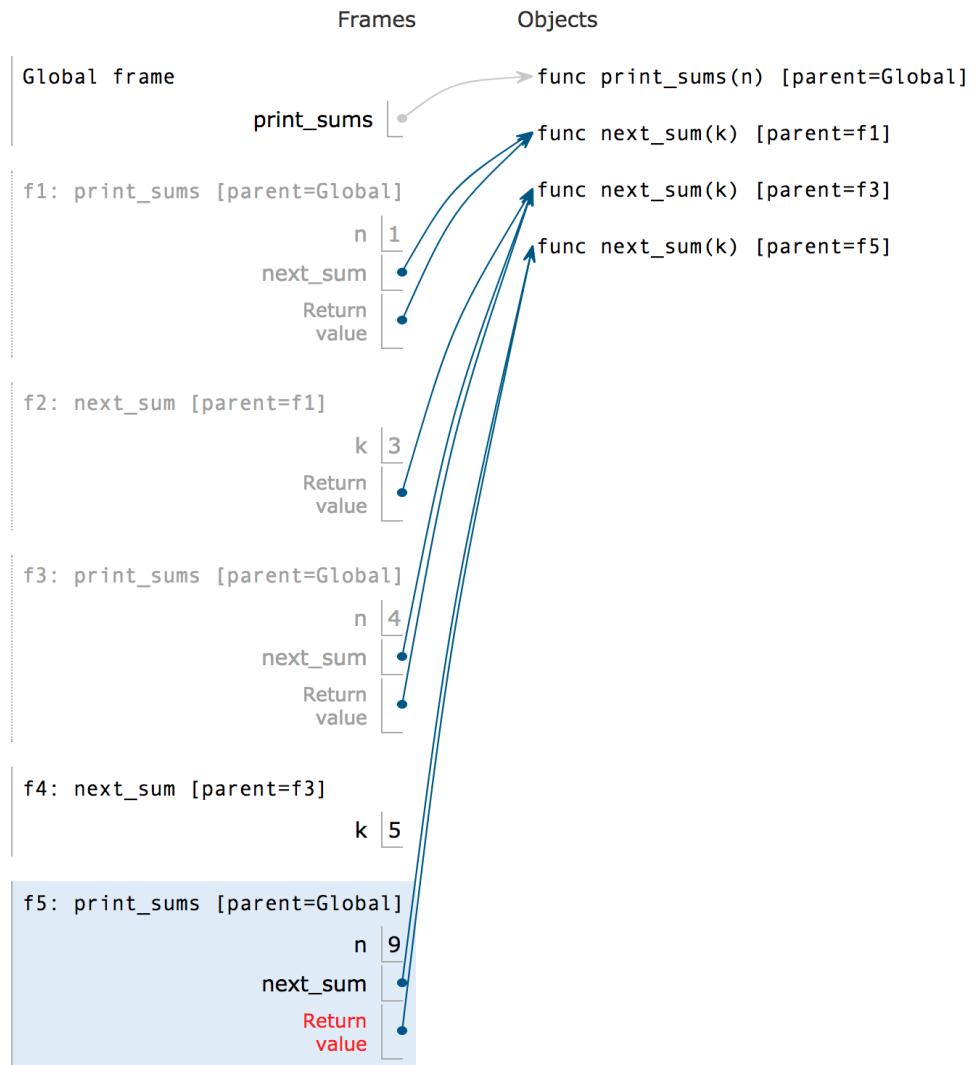
```
1 def print_sums(n):  
2     print(n)  
3     def next_sum(k):  
4         return print_sums(n+k)  
5     return next_sum  
6  
→ 7 print_sums(1)(3)(5)
```

`print_sums(1)(3)(5)` prints:

$$\begin{array}{r} 1 \\ 4 \\ 9 \end{array} \quad \begin{array}{r} (1 + 3) \\ (1 + 3 + 5) \end{array}$$

`print_sums(3)(4)(5)(6)` prints:

$$\begin{array}{r} 3 \\ 7 \quad (3 + 4) \\ 12 \quad (3 + 4 + 5) \\ 18 \quad (3 + 4 + 5 + 6) \end{array}$$



Example: Add Up Some Numbers (Fall 2016 Midterm 1 Question 5)

Implement `add_up`, which takes a positive integer `k`. It returns a function that can be called repeatedly `k` times, one integer argument at a time, and returns the sum of these arguments after `k` repeated calls.

```
def add_up(k):
    """Add up k numbers after k repeated calls.

    add_up(4)(10) returns a one-arg function & needs to remember 3 & 10

>>> add_up(4)(10)(20)(30)(40) # Add up 4 numbers: 10 + 20 + 30 + 40
100
"""
    assert k > 0
    def f(n):
        if k == 1:
            return n
        else: return lambda t: add_up(k - 1)(n + t)
    return f
```

add_up(4) returns a one-arg function & needs to remember the 4

Evaluates to a one-arg function that

Recursive Functions

(Demo)

Discussion Question: Factorial Two Ways

Rewrite fact(n) so that the result of fact(5) is computed using the following steps:

```
5 (1 * 5)
20 (1 * 5 * 4)
60 (1 * 5 * 4 * 3)
120 (1 * 5 * 4 * 3 * 2)
```

```
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
```

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)
```