Recursion

Announcements

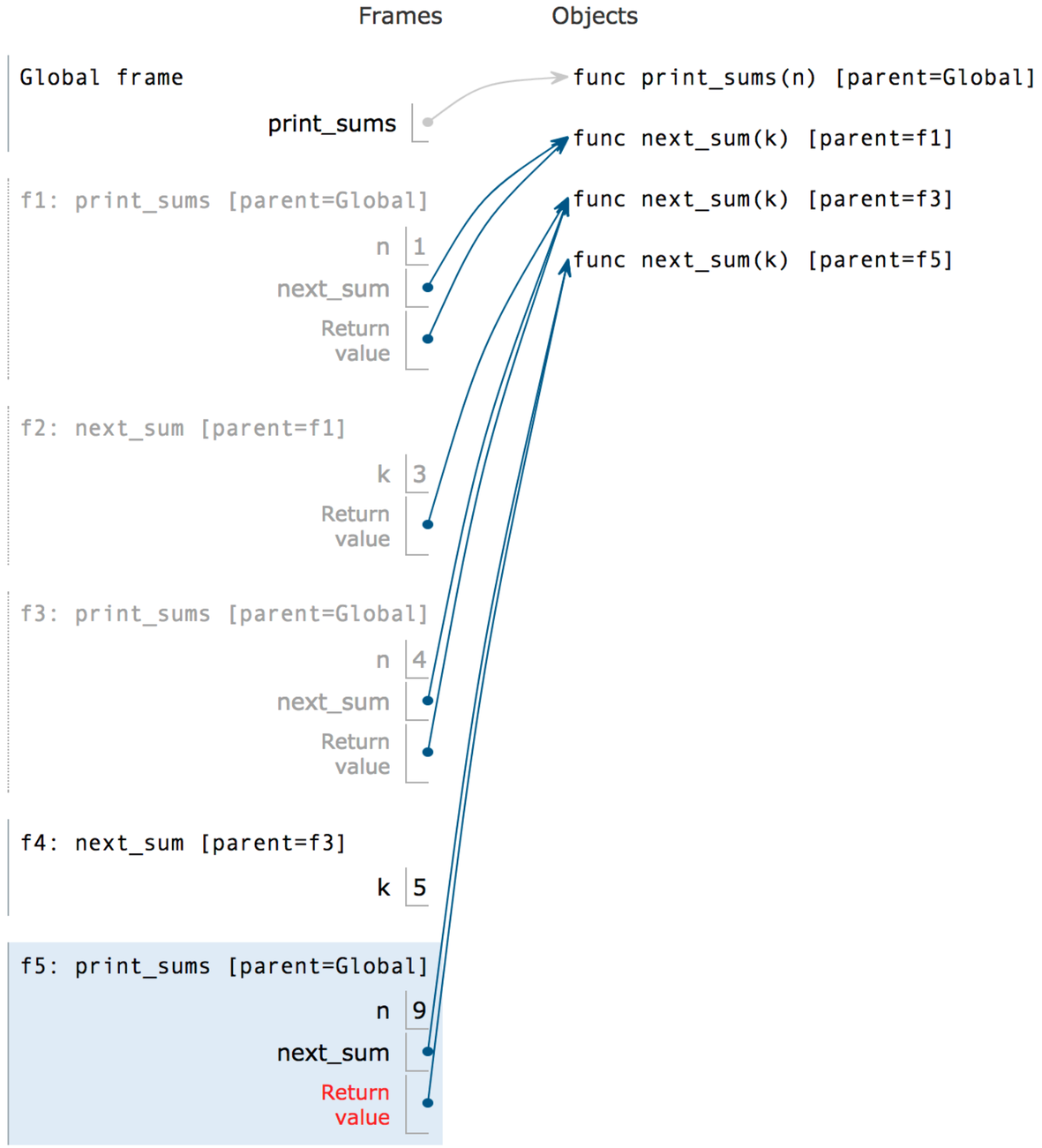
Self-Reference

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:
1
4  (1 + 3)
9  (1 + 3 + 5)

print_sums(3)(4)(5)(6) prints:
3
7  (3 + 4)
12 (3 + 4 + 5)
18 (3 + 4 + 5 + 6)
```



Example:

What is the last line that will be printed out?

```
1  def add_next(n):
2      print(n)
3      return lambda f: subtract_next(n + f)
4
5  def subtract_next(n):
6      print(n)
7      return lambda f: add_next(n - f)
8
9  add_next(2500)(500)(1000)(24)
```

Answer: 2024

These are functions, that return functions, that call each other!

https://pythontutor.com/cp/composingprograms.html#code=def%20add_next%28n%29%3A%0A%20%20%20%20print%28n%29%0A%20%20%20%20return%20lambda%20f%3A%20subtract_next%28n%20%2B%20f%29%0A%0Adef%20subtract_next%28n%29%3A%0A%20%20%20%20print%28n%29%0A%20%20%20%20return%20lambda%20f%3A%20add_next%28n%20-%20f%29%0A%0Aadd_next%282500%29%28500%29%281000%29%2824%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Recursive Functions

(Demo)

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

From Iteration to Recursion

Example: Boxes and Pyramids

Here's a simple function *boxes_iter(k)* that iteratively prints out k boxes in a line.
(a pair of square brackets, like: [])
Can we make it recursive?

```
def boxes_iter(k):
    """ prints out k boxes.
    >>> boxes_iter(4)
    [][][][]
    """
    while k > 0:
        print("[]", end="")
        k -= 1
    return
```

```
def boxes_r(k):
    """ prints out k boxes.
    >>> boxes_r(4)
    [][][][]
    """
    if k == 0:
        return
    else:
        print("[]", end="")
        boxes_r(k-1)
    return
```

base case

recursive case

https://pythontutor.com/cp/composingprograms.html#code=def%20boxes%28k%29%3A%0A%20%20%20%20%22%22%22prints%20out%20k%20boxes%3A%20%E2%80%9C%5B%5D%E2%80%9D.%0A%20%20%20%20%3E%3E%3E%20boxes%284%29%20%20%23%20prints%20out%204%20boxes%0A%20%20%20%20%5B%5D%5B%5D%5B%5D%5B%5D%0A%20%20%20%20%22%22%22%0A%20%20%20%20while%20k%20%3E%200%3A%0A%20%20%20%20%20%20%20%20print%28%22%5B%5D%22,%20end%3D%22%22%29%0A%20%20%20%20%20%20%20%20k%20-%3D%201%0A%20%20%20%20return%0A%0Aboxes%284%29&cumulative=true&curInstr=18&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Example: Boxes and Pyramids

Here's a function *pyramid* that prints out a *pyramid* of k height using one of the boxes functions we wrote on the last slide.

```
def pyramid_iter(k):
    """ prints out a pyramid of k height.
    >>> pyramid(4)
    []
    [][]
    [][][]
    [][][][]
    """
    i = 1;
    while i <= k:
        print(" ") # new line
        boxes_r(i)
        i += 1
    return
```

```
def pyramid(k):
    """ prints a pyramid of k height.
     >>> pyramid(4)
    []
    [][]
    [][][]
    [][][][]
    """
    if  k == 0:   # base case
        return
    else:
        pyramid(k-1)  # print a pyramid of k-1 height
        print("")     # skip a line,
        boxes_r(k)    # print out k boxes.
    return
```

How would we flip the pyramid upside-down?

https://pythontutor.com/cp/composingprograms.html#code=def%20boxes_r%28k%29%3A%0A%20%20%20%22%22%22prints%20k%20boxes%20in%20a%20line.%0A%20%20%20%3E%3E%3E%20boxes%284%29%20%20%0A%20%20%20%5B%5D%5B%5D%5B%5D%5B%5D%0A%20%20%20%22%22%22%0A%20%20%20if%20k%20%3D%3D%200%3A%20%0A%20%20%20%20%20%20%20return%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20print%28%22%5B%5D%22,%20end%3D%22%22%29%0A%20%20%20%20%20%20%20boxes_r%28k-1%29%0A%20%20%20%20return%20%0A%0Adef%20pyramid_r%28k%29%3A%0A%20%20%20%20%22%22%22%20recursively%20prints%20out%20a%20pyramid%20of%20k%20height.%0A%20%20%20%3E%3E%3E%20pyramid%284%29%20%20%0A%20%20%20%20%20%5B%5D%0A%20%20%20%20%20%5B%5D%5B%5D%0A%20%20%20%20%20%5B%5D%5B%5D%5B%5D%0A%20%20%20%20%20%5B%5D%5B%5D%5B%5D%5B%5D%0A%20%20%20%20%22%22%22%0A%20%20%20%20if%20k%3D%3D0%3A%0A%20%20%20%20%20%20%20return%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20pyramid_r%28k-1%29%0A%20%20%20%20%20%20%20print%28%22%22%29%0A%20%20%20%20%20%20%20boxes_r%28k%29%0A%20%20%20%20return%0A%0Apyramid_r%284%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Upside-Down Pyramids

In the iterative example, we must fiddle with variables.

```
def pyramid_iter(k):
    """ prints out a pyramid of k height.
    >>> pyramid(4)
    [][][][]
    [][][]
    [][]
    []
    """
    while k > 0:    # start with k boxes
        print(" ") # new line
        boxes_r(k)
        k -= 1      # decrement k
    return
```

In the recursive example, we can just switch the order of these two lines.

```
def pyramid(k):
    """ prints a pyramid of k height.
     >>> pyramid(4)
    [][][][]
    [][][]
    [][]
    []
    """
    if  k == 0:    # base case
        return
    else:
        boxes_r(k)     # print out k boxes.
        print("")      # skip a line,
        pyramid(k-1)   # print a pyramid of k-1 height
    return
```

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)
```