RECURSION, TREE RECURSION, AND ORDERS OF GROWTH

COMPUTER SCIENCE 61A

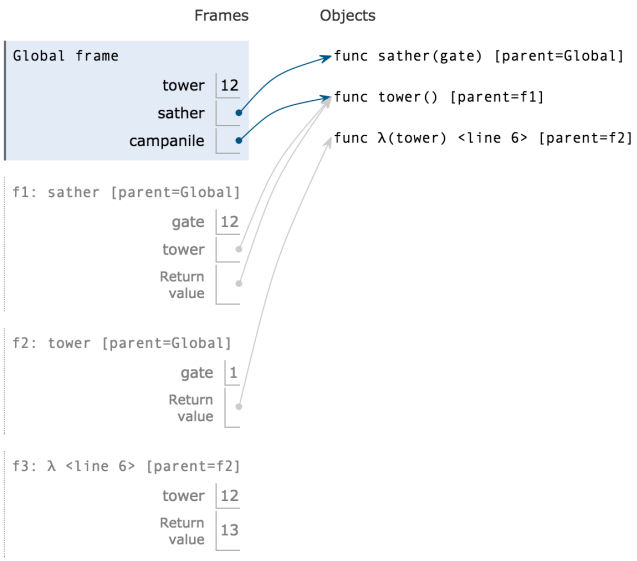
July 4 to July 10, 2015

1 Higher Order Functions

1. Draw an environment diagram for the following code.

```
tower = 12

def sather(gate):
    def tower():
        gate = 1
        return lambda tower: tower + gate
    return tower

campanile = sather(tower)
campanile()(12)
```

Solution:



2. Implement `mystery`, a function that passes the following doctests:

```
def mystery(func, n):
    """
    >>> from operator import add, mul
    >>> a = mystery(add, 3)
    >>> a(4)
    7
    >>> a(12)
    15
    >>> b = mystery(lambda x, y: x*x + y, 4)
    >>> b(5)
    21
    >>> b(7)
    23
    """
```

Solution:

```
def mystery(func, n):
    def inner(y):
        return fn(n, y)
    return inner
```

2 Recursion

1. What is a recursive function?

   Solution: A recursive function is a function that calls itself in its body, either directly or indirectly.

2. What are 3 important components that all recursive functions have?

   Solution:

   1. One or more base cases
   2. Way(s) to make the problem smaller
   3. One or more recursive cases

3. What is a tree recursive function? How is it different from a linearly recursive function?

   Solution: Tree recursive functions are functions that makes more than one recursive call. This is different from linearly recursive functions (such as factorial), because linearly recursive functions make exactly one recursive call in the recursive case.

4. Do the following recursive functions work as intended? If not, find the bug and fix it.

```
def find_digit(number, digit):
    """Return true if the digit is included in the given
    number. Return false otherwise.

    >>> find_digit(4, 4)
    True
    >>> find_digit(4356, 4)
    True
    >>> find_digit(4356, 8)
    False
    >>> find_digit(3, 4)
    False
    """
    if number % 10 == digit:
        return True
    else:
        return find_digit(number // 10, digit)
```

Solution: This function is missing another base case:

```
    ...
    elif number < 10:
        return False
    ...
```

```
def sum_digits(number):
    """Return the sum of all digits in a number.
    >>> sum_digits(4)
    4
    >>> sum_digits(43)
    7
    >>> sum_digits(123456789)
    45
    """
    if number < 10:
        return number
    else:
        return sum_digits(number % 10) + number // 10
```

Solution: The recursive call is taking in the wrong argument. The recursive call to `sum_digits` is being given only the ones digit, but it should take in the entire number *without* the ones digit.

```
    ...
        return sum_digits(number // 10) + number % 10
```

5. Implement `sorted_digits(n)`, a function that takes in a number `n` and returns True if the digits of `n` are increasing from right to left.

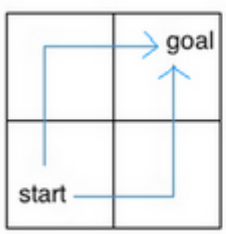
```
def sorted_digits(number):
    """Return True if the digit is in increasing order from
    rightmost digit to leftmost digit. (Consecutive digits
    that are the same are allowed.) Also return True if it
    has only one digit. Return False otherwise.

    >>> sorted_digits(2)
    True
    >>> sorted_digits(22222)
    True
    >>> sorted_digits(9876543210)
    True
    >>> sorted_digits(9087654321)
    False
    """
```

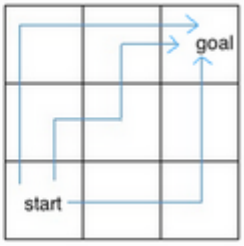
Solution:

```
    ones_digit = number % 10
    rest =  number // 10
    if rest == 0:
        return True
    elif ones_digit > rest % 10:
        return False
    else:
        return sorted_digits(rest)
```

6. Implement `path(n)`, which returns the number of paths from one corner of an n x n grid to the opposite corner.

   Consider an insect in an N by N grid. The insect starts at the bottom left corner, (0, 0), and wants to end up at the top right corner, (N-1, N-1). The insect is only capable of moving right or up. Write a function paths that takes a grid length and width and returns the number of different paths the insect can take from the start to the goal. (There is a closed-form solution to this problem, but try to answer it procedurally using recursion.)





   For example, the 2 by 2 grid has a total of two ways for the insect to move from the start to the goal. For the 3 by 3 grid, the insect has 6 different paths (only 3 are shown above).

```
def paths(n):
    """Return the number of paths from one corner of an
    N by N grid to the opposite corner.

    >>> paths(2)
    2
    >>> paths(3)
    6
    >>> paths(10)
    48620
    """
```

Solution:

```
    def helper(row, col):
        if row == 1 or col == 1:
            return 1
        return helper(row-1, col) + helper(row, col-1)
    return helper(n, n)
```

3 Orders of growth

1. Write down the orders of growth for the following functions in terms of n.

```
def a(n):
    if n <= 0:
        return 1
    return 1 + a(n // 2)
```

Solution: $O(\log(n))$

```
def loopy(n):
    result = 0
    while n > 0:
        result += n
        n -= 1
    return result
```

Solution: $O(n)$