Recursion and Tree Recursion

CS 61A Group Mentoring

July 2, 2018

1 Recursion

Every Recursive function has three things.

1. One or more base cases
2. One or more ways to break the problem down into a smaller problem
    - E.g. Given a number as input, we need to break it down into a smaller number
3. Solve the smaller problem recursively; from that, form a solution to the original problem

1. What is wrong with the following function? How can we fix it?

```
def factorial(n):
    return n * factorial(n)
```

Solution: There is no base case and the recursive call is made on the same `n`.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

2. Complete the definition for `all_true`, which takes in a list `lst` and returns `True` if there are no False-y values in the list and `False` otherwise. Make sure that your implementation is recursive.

```
def all_true(lst):
    """
    >>> all_true([True, 1, "True"])
    True
    >>> all_true([1, 0, 1])
    False
    >>> all_true([])
    True
    """
```

Solution:

```
    if not lst:
        return True
    elif not lst[0]:
        return False
    else:
        return all_true(lst[1:])
```

3. Write a function `is_sorted` that takes in an integer `n` and returns true if the digits of that number are nondecreasing from right to left.

```
def is_sorted(n):
    """
    >>> is_sorted(2)
    True
    >>> is_sorted(22222)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(9087654321)
    False
    """
```

Solution:

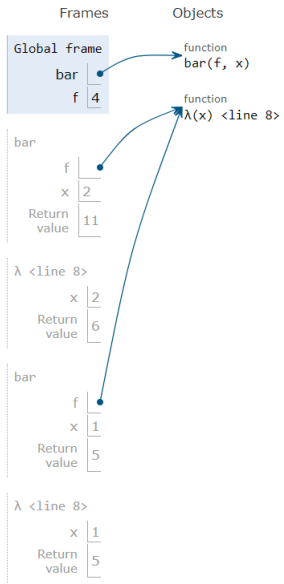
```
    right_digit = n % 10
    rest =  n // 10
    if rest == 0:
        return True
    elif right_digit > rest % 10:
        return False
    else:
        return is_sorted(rest)
```

4. Draw the environment diagram that results from running the code.

```
def bar(f, x):
    if x == 1:
        return f(x)
    else:
        return f(x) + bar(f, x - 1)

f = 4
bar(lambda x: x + f, 2)
```

Solution: Output: 3



https://goo.gl/kR2n9M

5. Write a function that takes as input a number, `n`, and a list of numbers, `lst`, and returns true if we can find a subset of `lst` that sums up to `n`.

Solution:

```
def add_up(n, lst):
    """
    >>> add_up(10, [1, 2, 3, 4, 5])
    True
    >>> add_up(8, [2, 1, 5, 4, 3])
    True
    >>> add_up(-1, [1, 2, 3, 4, 5])
    False
    >>> add_up(100, [1, 2, 3, 4, 5])
    False
    """
    if n == 0:
        return True
    if lst == []:
         return False
    else:
        first, rest = lst[0], lst[1:]
        return add_up(n - first, rest) or add_up(n, rest)
```