CS 61A
Fall 2019

Recursion and Tree Recursion

Guerrilla Section 1: September 27, 2019

1 Recursion and Tree Recursion

Questions

1.1 What are three things you find in every recursive function?

(1) Base case
(2) divide into small thing
(3) call themselves.

1.2 When you write a Recursive function, you seem to call it before it has been fully defined. Why doesn't this break the Python interpreter?

Because when you call a function, which will create a frame,

1.3 Below is a Python function that computes the nth Fibonacci number. Identify the three things it contains as a recursive function (from 1.1).

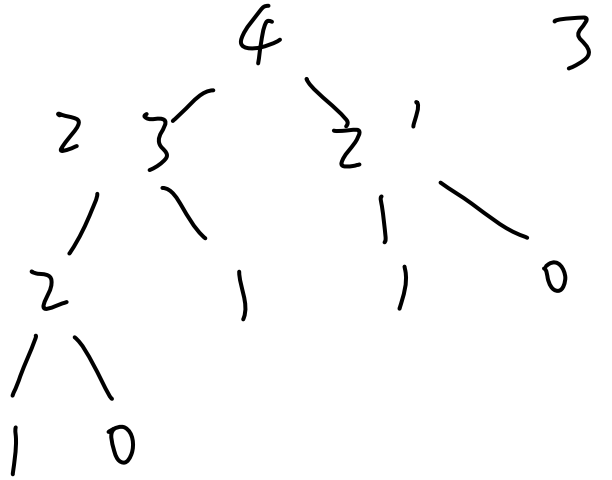
```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

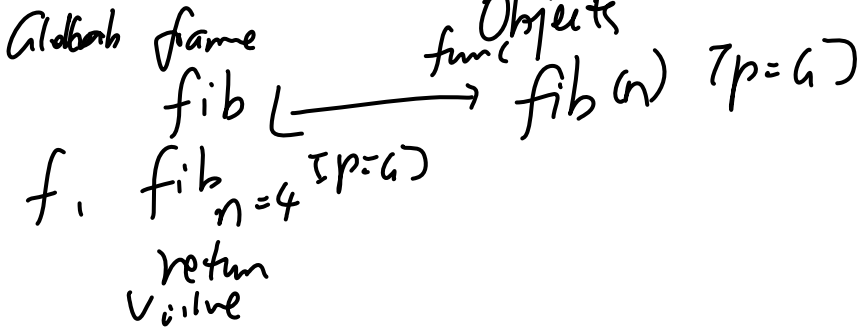
Base case: if n==0
n==1:

divide: fib(n-1) + fib(n-2)

Call themselves: return

1.4 With the definition of the Fibonacci function above, draw out a diagram of the recursive calls made when **fib(4)** is called.





1.5 What does the following function **cascade2** do? What is its domain and range?

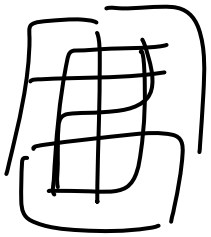
```
def cascade2(n):
    print(n)
    if n >= 10:
        cascade2(n//10)
        print(n)
```

print anything

domain: $-\infty$ ~ ∞

range: 10 ~ ∞

1.6 Consider an insect in an `M` by `N` grid. The insect starts at the bottom left corner, `(0, 0)`, and wants to end up at the top right corner `(M-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.)



```
def paths(m, n):
    """
    >>> paths(2, 2)
    2
    >>> paths(117, 1)
    1
    """
    if n==1 and m==1:
        return 0
    if n == 1:
        return 1
    if m == 1:
        return 1
    return paths(m-1,n) + paths(m, n-1)
```

1.7 Write a procedure `merge(s1, s2)` which takes two sorted (smallest value first) lists and returns a single list with all of the elements of the two lists, in ascending order. Use recursion.

Hint: If you can figure out which list has the smallest element out of both, then we know that the resulting merged list will have that smallest element, followed by the merge of the two lists with the smallest item removed. Don't forget to handle the case where one list is empty!

```
def merge(s1, s2):
    """ Merges two sorted lists
    >>> merge([1, 3], [2, 4])
    [1, 2, 3, 4]
    >>> merge([1, 2], [])
    [1, 2]
    """
    if len(s1) == 0:
        return s2
    if len(s2) == 0:
        return s1
    if len(s1) == 0 and len(s2) == 0:
        return []
    if s1[0] > s2[0]:
        return merge(s1, s2[1:]) + s2[0]
    elif s1[0] < s2[0]:
        return s1[0] + merge(s1[1:], s2)
    else:
        return s1[0] + s2[0] + merge(s1[1:], s2[1:])
```

1.8 Mario needs to jump over a sequence of Piranha plants, represented as a string of dashes (no plant) and P's (plant!). He only moves forward, and he can either step (move forward one place) or jump (move forward two places) from each position. How many different ways can Mario traverse a level without stepping or jumping into a Piranha plant? Assume that every level begins with a dash (where Mario starts) and ends with a dash (where Mario must end up):

Hint: You can get the ith character in a string 's' by using 's[i]'. For example,

```
>>> s = 'abcdefg'
>>> s[0]
'a'
>>> s[2]
'c'
```

You can find the total number of characters in a string with the built-in'len' function:

```
>>> s = 'abcdefg'
>>> len(s)
7
>>> len('')
0
```

```
def mario_number(level):
    """Return the number of ways that Mario can perform a sequence of steps
    or jumps to reach the end of the level without ever landing in a Piranha
    plant. Assume that every level begins and ends with a dash.

    >>> mario_number('-P-P-')   # jump, jump
    1
    >>> mario_number('-P-P--')   # jump, jump, step
    1
    >>> mario_number('--P-P-')  # step, jump, jump
    1
    >>> mario_number('---P-P-') # step, step, jump, jump or jump, jump, jump
    2
    >>> mario_number('-P-PP-')  # Mario cannot jump two plants
    0
    >>> mario_number('----')    # step, jump ; jump, step ; step, step, step
    3
    >>> mario_number('----P----')
    9
    >>> mario_number('---P----P-P---P--P-P----P-----P-')
    180
    """
```