Guerrilla Section 2: Higher Order Functions, Sequences, & Recursion/Tree Recursion

Solutions

Instructions

Form a group of 3-4. Start on Question 0. Check off with a lab assistant when everyone in your group understands how to solve Question 0. Repeat for Question 1, 2, etc. **You're not allowed to move on from a question until you check off with a tutor.** You are allowed to use any and all resources at your disposal, including the interpreter, lecture notes and slides, discussion notes, and labs. You may consult the lab assistants, **but only after you have asked everyone else in your group. The purpose of this section is to have all the students working together to learn the material. The questions defined as EXTRA are optional, make sure you finish all the non-optional ones before you attempt the extra ones.**

Higher Order Functions

Question 3

Write a `make_skipper`, which takes in a number `n` and outputs a function. When this function takes in a number `x`, it prints out all the numbers between 0 and `x`, skipping every `n`th number (meaning skip any value that is a multiple of `n`).

```
>>> a = make_skipper(2)
>>> a(5)
1
3
5
def make_skipper(n):


    def skipper(x):
        for i in range(x+1):
          if i % n != 0:
            print(i)
      return skipper
```

EXTRA: Question 4

Write make_alternator which takes in two functions, f and g, and outputs a function. When this function takes in a number x, it prints out all the numbers between 1 and x, applying the function f to every odd-indexed number and g to every even-indexed number before printing.

“””

```
>>> a = make_alternator(lambda x: x * x, lambda x: x + 4)
>>> a(5)
1
6
9
8
25
>>> b = make_alternator(lambda x: x * 2, lambda x: x + 2)
>>> b(4)
2
4
6
6
```

“””

```
def make_alternator(f, g):
  def alternator(n):
    i = 1
    while i <= n:
      if i % 2 == 1:
        print(f(i))
      else:
        print(g(i))
      i += 1
  return alternator

# Alternatively
>>> def make_alternator(f, g):
...   def alternator(n):
...     for i in range(1, n+1):
...       print(f(i)) if i % 2 == 1 else print(g(i))
...   return alternator
```

STOP!

Don’t proceed until everyone in your group has finished and understands all exercises in this section, and you have gotten checked off!

Recursion

Question 0

a) What are three things you find in every recursive function?

1. One or more Base Cases

2. Way(s) to make the problem into a smaller problem of the same type (so that it can be solved recursively).

3. One or more Recursive Cases that solve the smaller problem and then uses the solution the smaller problem to solve the original (large) problem

b) 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? Explain in haiku if possible.

When you define a function, Python does not evaluate the body of the function.

Python does not care
about a function's body
until it is called

Question 1

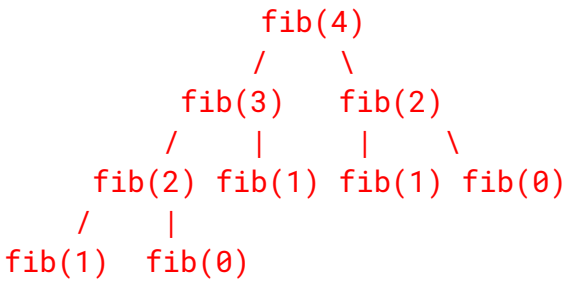
Hint: Domain is the type of data that a function takes in as argument. The Range is the type of data that a function returns.

E.g. the *domain* of the function `square` is numbers. The *range* is numbers.

a) Here is a Python function that computes the nth fibonnaci number. What's it's domain and range? Identify those three things from **Q0a**:

```
def fib(n): Domain is integers, Range is integers
  if n == 0: # base case
    return 0
  elif n == 1: another base case
    return 1
  else: # ONE recursive CASE with TWO recursive CALLS
    return fib(n-1) + fib(n-2) # reducing the problem
```

Write out the recursive calls made when we do `fib(4)` (this will look like an upside down tree).



b) What does the following `cascade2` do?

Takes in a number n and prints out n, n excluding the ones digit, n excluding the tens digit, n excluding the hundreds digit, etc, then back up to the full number

```
def cascade2(n):
    """Print a cascade of prefixes of n."""
    print(n)
    if n >= 10:
        cascade2(n//10)
        print(n)
```

What is the domain and range of `cascade2`? Identify the three things from Q0a:

```
def cascade2(n): Domain is integers, Range is None
    """Print a cascade of prefixes of n."""
    print(n) # Base case is when n < 10
    if n >= 10: # recursive case
        cascade2(n//10) # reducing the problem
        print(n)
```

c) Describe what does each of the following functions mystery and fooply do.

```
>>> def mystery(n):
...   if n == 0:
...         return 0
...   else:
...         return n + mystery(n - 1)
```

Sums integers up to n: 1 + 2 + 3 + 4 + 5 +…+ n

```
>>> def foo(n):
...   if n < 0:
...         return 0
...   return foo(n - 2) + foo(n - 1)

>>> def fooply(n):
...   if n < 0:
...         return 0
...   return foo(n) + fooply(n - 1)
```

Foo returns the nth fibonacci number
fooply returns the sum of first n fibonacci numbers

STOP!

Don’t proceed until everyone in your group has finished and understands all exercises in this section, and you have gotten checked off!

Question 2

Mario needs to jump over a series of Piranha plants, represented as a list of 0's and 1's. Mario only moves forward and can either step (move forward one space) or jump (move forward two spaces) 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 1 (where Mario starts) and ends with a 1 (where Mario must end up).

```
def mario_number(level):
    """
    Return the number of ways that Mario can traverse the level,
    where Mario can either hop by one digit or two digits each turn.
    A level is defined as being an integer with digits where a 1 is
    something Mario can step on and 0 is something Mario cannot step
    on.
    >>> mario_number(10101)  #Hops each turn: (1, 2, 2)
    1
    >>> mario_number(11101) #Hops each turn: (1, 1, 1, 2), (2, 1, 2)
    2
    >>> mario_number(100101) #No way to traverse through level
    0
    """
    if level == 1:
        return 1
    elif level % 10 == 0:
        return 0
    else:
        return mario_number(level // 10) + mario_number((level // 10) // 10)
```

EXTRA Challenge: Question 3

Implement the combine function, which takes a non-negative integer n, a two-argument function f, and a number result. It applies f to the first digit of n and the result of combining the rest of the digits of n by repeatedly applying f (see the doctests). If n has no digits (because it is zero), combine returns result. Assume n is non negative.

```
def combine (n , f , result ):

"""
Combine the digits in n using f.
>>> combine (3, mul, 2) #mul (3, 2)
6
>>> combine (43, mul, 2) # mul (4, mul(3, 2))
24
>>> combine (6502, add, 3) # add (6, add(5, add(0, add(2, 3)))
16
>>> combine (239, pow, 0) # pow (2, pow(3, pow(9, 0)))
8
"""

if n == 0:
      return result
else:
      return ___combine ( n // 10 , f , f ( n % 10 , result ))____
```

STOP!

Don’t proceed until everyone in your group has finished and understands all exercises in this section, and you have gotten checked off!