**This document: http://tinyurl.com/kjavmfh**

**Question 0**

**a)** Explain Recursion. If it doesn't take too long, write your explanation in haiku.

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

1.

2.

3.

**c)** 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.

**e)** What does the following code do? Describe it in English. Hint: diagram the function in terms of the base cases, the recursive cases, and reducing the problem.

def mystery(lst, toggle=False):

if lst == []:

return []

elif toggle:

return [lst[0]] + mystery(lst[1:], False)

else:

return mystery(lst[1:], True)

**f)** What's the base case? What are the recursive cases?

**g)** What would mystery([1, 2, 3, 4]) output?

**h)** How about mystery([2, 3, 4, 5], True)?

**i)** What does toggle do?

**j)** Could this function work with strings? E.g. mystery(“I am a string”).

**Question 1**

**Hint:** Domain is the type of data (integers, strings, etc) the 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)** What is the mathematical definition of a fibonacci number?

Here is a Python function that computes the nth fibonnaci number. What's it's domain and range? Identify those three things from **Q0b**:

def fib(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fib(n-1) + fib(n-2)

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?

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? Identify the three things from Q0b:

**Question 2**

**a)** What's wrong with the following code? Identify all the mistakes and fix them.

def multiply(x, y):

""" Multiplies two numbers together without using the \* operator or mul

>>> multiply(3, 4)

12

"""

if x == 0:

return 1

else:

return y + multiply(x, y)

**b)** What does sum\_eo\_list do?

>>> sum\_eo\_list([1, 2, 3, 4, 5, 6, 7])

12

>>> sum\_eo\_list([“These”, 2, “elements”, 4, “are”, 6, “irrelevant”])

12

>>> sum\_eo\_list([1])

0

What's wrong with the following code? Identify all the things and fix them.

def sum\_eo\_list(lst, sum=False):

first, rest = lst[0], lst[1:]

if sum:

return first + sum\_eo\_list(rest, False)

if lst == []:

0

else:

return sum\_eo\_list(rest, True)

**Question 3**

**Hint:** Remember that % (mod) is a useful operator.

Write recursive fizzbuzz. Hint: first identify the three things from **Q0b**.

def fizzbuzz(n):

“”” Prints everything from 1 to n. If it’s divisible by 3, instead it

prints ‘fizz!’. If it’s divisible by 5, it prints ‘buzz!!’. If it’s

divisible by both, it prints ‘fizzbuzz!!!’

“””

**Question 4**

Write recursive foobar: Hint: first identify the three things from **Q0b**.

>>> foobar(0)

"foo"

>>> foobar(1)

"foobar"

>>> foobar(2)

"foobarbar"

>>> foobar(3)  
"foobarbarfoo"

>>> foobar(4)  
"foobarbarfoobar"

>>> foobar(14)

"foobarbarfoobarbarfoobarbarfoobarbarfoobarbar"

**Question 5**

Write deep\_lst\_sum. Hint: first identify the three things from **Q0b**.

>>> deep\_lst\_sum([6, 2, [8, 4, 5], [9, [5]]])

39

It's dangerous to go alone, take this!

def is\_list(e):

""" tests if an element is a list.

>>> is\_list([1, 2])

True

>>> is\_list(4)

False

"""

return type(e) == list

def is\_int(e):

""" tests if an element is an integer

>>> is\_int([1, 2])

False

>>> is\_int(4)

True

"""

return type(e) == int

**Question 6**

**a)** Write insert\_every. Hint: first identify the three things from **Q0b**.

>>> insert\_every(1, [[2], [2, 3]])

[[1, 2], [1, 2, 3]]

**b)** Write powerset. Hint: first identify the three things from **Q0b**. Hint2: use insert\_every

>>> powerset([1, 2, 3])  
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]

**Question 7**

Write longest\_inc\_subseq. Hint: first identify the three things from **Q0b**. Hint2: use a helper!

>>> longest\_inc\_subseq([1, 2, 3])

[1, 2, 3]

>>> longest\_inc\_subseq([2, 1, 2, 3])

[1, 2, 3]

>>> longest\_inc\_subseq([2, 1, 2, 3, 1])

[1, 2, 3]

>>> longest\_inc\_subseq([1, 8, 9, 2, 41, 10, 19, 2, 1])

[1, 8, 9, 10, 19]

>>> longest\_inc\_subseq([1, 1, 12, 3, 8, 1, 9])

[1, 3, 8, 9]

**Question 8**

Write middle without using len. Hint: first identify the three things from **Q0b**. Hint2: write a helper function.

>>> middle([1, 2, 3, 4, 5])

3

>>> middle([1, 2, 3, 4])

2

>>> middle([1])

1

>>> middle([])

False

Challenge (optional): did your solution involve counting the number of elements? There's another way. Hint: use [1:] and [2:]

**Question 9**

**a)** Write add2up

>>> add2up(10, [1, 2, 3, 4, 5])

False

>>> add2up(8, [1, 2, 3, 4, 5]) # 3 + 5 = 8

True

>>> add2up(3, [1, 2, 3, 4, 5]) # 1 + 2 = 3

True

>>> add2up(2, [1, 2, 3, 4, 5]) # ? + ? = 2

False

**b)** Write addup. Hint: Tree recursion!

>>> addup(10, [1, 2, 3, 4, 5])

True

>>> addup(8, [1, 2, 3, 4, 5])

True

>>> addup(3, [1, 2, 3, 4, 5])

True

>>> addup(2, [1, 2, 3, 4, 5])

True

>>> addup(-1, [1, 2, 3, 4, 5])

False

>>> addup(100, [1, 2, 3, 4, 5])

False

>>> addup(11, [2, 2, 2, 2, 2, 2])

False

## Challenge Questions (optional)

def edit\_distance(a, b):  
 """  
 Find the edit distance between two strings.

Edit distance is defined as the minimum number of insertions,

deletions, and character changes it takes to get from one string

to another.  
  
 >>> edit\_distance("saturday", "sunday")  
 3  
 >>> edit\_distance("kitten", "sitting")  
 3  
 >>> edit\_distance("", "this is really weird")  
 20  
 >>> edit\_distance("book", "back")  
 2  
 >>> edit\_distance("hello world", "yo man")  
 9  
 """

def knapsack(weights, profits, weight\_capacity):  
 """  
 You are a thief, and you have a bunch of items in front of you.  
 Each item (represented by a certain index) has a corresponding

weight and profit.  
   
 You have a weight capacity than you can not exceed (so you can't

take all the items), what is the maximum profit you can gain

given the items and the capacity you can carry?  
   
 >>> knapsack([23, 31, 29, 44, 53, 38, 63, 85, 89, 82], \

[92, 57, 49, 68, 60, 43, 67, 84, 87, 72], 165)  
 309   
 >>> knapsack([1, 1, 1], [1, 2, 3], 3)  
 6  
 >>> knapsack([1, 2, 2], [1, 2, 3], 3)  
 4  
 >>> knapsack([1, 2, 2], [1, 3, 3], 2)  
 3 """

def hanoi(n\_rings):  
 """  
 Calculate the minimum number of moves to solve towers of  
 hanoi. Do it recursively.  
  
 >>> hanoi(3)  
 7  
 >>> hanoi(8)  
 255  
 >>> hanoi(1)  
 1  
 >>> hanoi(5)  
 31  
 """

def is\_maze\_solvable(grid):  
 """  
 Determine if you can get from the top left corner (0, 0) of the  
 grid to the bottom right corner (length-1, length-1)  
  
 >>> grid = \  
 [[0, 0, 0, 0, 0, 1], \  
 [1, 1, 0, 0, 0, 1], \  
 [0, 0, 0, 1, 0, 0], \  
 [0, 1, 1, 0, 0, 1], \  
 [0, 1, 0, 0, 1, 0], \  
 [0, 1, 0, 0, 0, 2]]  
 >>> is\_maze\_solvable(grid)  
 True  
 >>> grid = \  
 [[0, 0, 0, 0, 0, 1], \  
 [1, 1, 0, 0, 0, 1], \  
 [0, 0, 0, 1, 0, 0], \  
 [0, 1, 1, 0, 0, 1], \  
 [0, 1, 0, 0, 1, 0], \  
 [0, 1, 0, 0, 1, 2]]  
 >>> is\_maze\_solvable(grid)  
 False  
 """

def min\_to\_palindrome(start\_string):  
 """  
 Determine the minimum number of insertions and changes  
 required to convert start\_string into a palindrome.  
 (A palindrome is a string that is the same as its reverse.  
 E.g: racecar)  
  
 >>> min\_to\_palindrome("racecar")  
 0  
 >>> min\_to\_palindrome("abc")  
 1  
 >>> min\_to\_palindrome("happ")  
 2  
 >>> min\_to\_palindrome("madama")  
 1  
 >>> min\_to\_palindrome("abcdefghijk")  
 10  
 """