Cheatsheet

Properties of Tian Xiao

Time and Space Complexity (What is n?)

Time O(n): Explain what is n and which loop loops for n times.

Space O(1): Because there is no deferred operation or new object being created every iteration.

Tree: Time O(no. of leaves); Space O(depth)

Slicing: Time O(n); Space O(n)

Tuple Addition: Time O(n); Space O(n). n = len(tpl1) + len(tpl2).

Equality in Identity (is)

"a is b" is True only if the "=" assigns the same integer/boolean/string/variable to a and b.

Type of Errors

Syntax Error: Error in the syntax

Type Error: (1) Calling function with incorrect number of inputs (2) Unsupported operation symbol

Index Error: Sequence index out of range

Recursion Error/Infinite Loop: Maximum depth exceeded for recursion/iteration

Recursive Functions

Write a Recursive Function

1. Find the terminating condition.
2. Find f(n) in terms of f(n - 1).
3. The remaining part seems very easy.

Coin Change

```
def cc(amount, d):
    if amount == 0:
        return 1
    elif amount < 0 or d == 0:
        return 0
    else:
        return cc(amount - max_value) +\
               cc(amount, d - 1)
```

Hanoi

```
def hanoi(n, src, dst, aux):
    if n == 1:
        return ((src, dst),)
    else:
        return hanoi(n - 1, src, aux, dst)\
             + ((src, dst),)\
             + hanoi(n - 1, aux, dst, src)
```

Higher Order Functions

Lambda

lambda x: f(x) altogether is a function.

General Rule (foobar questions)

1. From left to right
2. Bracket first

Fold (fold(op, f, n))

How to find op, f, n?

1. Determine the type of output of f by observing the base case (e.g. f(0)).
2. Determine the type of op based on the output of f (e.g. Boolean → and/or).
3. The remaining part seems very easy.

Tuple Operations

Use Tuple to Represent Data

No. of elements + Meaning of each element

Tuple Slicing

Slicing always returns a tuple (never index error). (e.g. a = (); a[2:] → ())

Enumerate Leaves

```
def is_leaf(tree):
    return type(tree) != tuple

def enumerate_leaves(tree):
    if tree == ():
        return 0
    elif is_leaf(tree):
        return (tree,)
    else:
        return enumerate(tree[0]) +\
               enumerate(tree[1:])
```

Map

```
def map(f, tpl):
    if tpl == ():
        return ()
    else:
        return f(tpl[0]) + map(f, tpl[1:])
```

Filter

```
def filter(p, tpl):
    if p(tpl[0]):
        return tpl[0] + filter(p, tpl[1:])
    else:
        return filter(p, tpl[1:])
```

An element remains if it matches predicate.

Good luck!