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).

i in seq: Time O(n) for tuple and list; O(1) for dictionary. n = len(seq).

len(seq): Time O(1)

seq[key]: Time O(1)

max/min(seq, key): Time O(n). n = len(seq).

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

SyntaxError: Error in the syntax

Attribute Error: Attribute assignment or reference failed

TypeError: (1) Calling function with incorrect number of inputs (2) Unsupported operation symbol (3) Iterate an object not iterable

IndexError: Sequence/pop index out of range

RecursionError/Infinite Loop: Maximum depth exceeded for recursion/iteration

ValueError: (1) Remove something not in a list (2) Index something not in a list

KeyError: key not found in a dictionary

ZeroDivisionError: Division by zero

Error Raising

Try Except

```
try:
    <statement 1> # raise first error found
except Error1:
    <statement 2> # run if Error1 found
except:
    <statement 3> # generic error
else:
    <statement 4> # no error raised
finally:
    <statement 5> # run anyway
```

User-defined Error

```
def MyError(Exception):
    pass
```

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

```
     input   output
       ↖      ↗
lambda x: f(x)
```

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 element + 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

```
map(<mapping function>, seq) # generator
```

Filter

```
filter(<pred function>, seq) # generator
```

An element remains if it matches predicate.

List Operations (# Time Complexity)

.append(x): append an item # O(1)

.clear(): clear everything in a list # O(1)

.count(x): count the number of x # O(n)

.extend(lst): extend a list # O(k)

.index(x): return index of first x # O(n)

.insert(i, x): insert x at position i; append x if i exceeds len(lst) # O(n)

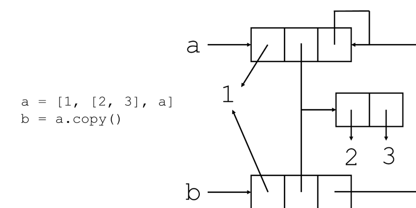
.pop(*i): remove and return lst[i] # O(n); remove and return lst[-1] if i not given # O(1)

.remove(x): remove first x # O(n)

.sort(key, reverse): sort a lst # O(nlogn)

sorted(lst, key, reverse): return a sorted lst # O(nlogn)

.copy(): return a shallow copy # O(n)



Dictionary Operations (# Time Complexity)

.keys(): return an iterable of keys # O(1)

.values(): return an iterable of values # O(1)

.items(): return an iterable of key-value pairs # O(1)

.clear(): clear everything in a dictionary # O(1)

.get(key, *value): get the value of the key; return <value> if key does not exist (default None) # O(1)

del dic[key]: delete a key in a dictionary # O(1)

.pop(k): remove and return dic[k] # O(1)

.update(dic): extend a dictionary # O(k)

.copy(): return a shallow copy # O(n)

Arbitrary Arguments

Unpack Variables

```
def f(*args):
    args # (arg1, arg2, ...)
```

Object-oriented Programming

Use Class to Represent Data

No. of property + Meaning of each property

Multiple Inheritance

1. From left to right.
2. From sub to super.
3. The remaining part seems very easy.

Dynamic Programming and Memoization

Working Condition: DP and memoization will improve performance when there are repetitions in the computation.

Good luck!

42 and the Meaning of Life