Built- in Functions:

Purpose	Example	
Ceiling	`math.ceil(1.5) = 2`	Round up its argument
Floor (//)		Round down to integer
Backslash (\)	`print('That's') XXX` `print('That\'s')`	Escapes normally special characters
Import	`from math import *` `import math`	(good if redefining previously defined functions e.g. max)
Round off to 3 d.p.	`round(ans,3)`	
Range	`range(start, stop, step)`	By default: `start`: 0 `step`: 1
Iteration	`continue`	Continues the next iteration of the loop/ next value
	`break`	Break out of the loop
Lambda	`(lambda x: x+1)(5) = 6`	The lambda function is like f.

Random	`from random import *`
	
	`random() = 0.6119987281172101`	

gcd	`from fractions import gcd`
	
	`g = gcd(n, d)`	

Reading a file	`input = open('inputfilename.txt', 'r')`
	
	`some_line = input.readline()`
		Opens up file for reading & reads line by line.
		End of file is empty string.

Writing to a file	`output = open('inputfilename.txt', 'w')`
	
	`output.write('HELLO WORLD')`	

Others:		
For fold	`op = lambda x, y: y/x`	If want division as op

Map	`def map(fn, seq):`
	`  if seq == ():`
	`  return ()`
	`  else:`
	`  return(fn(seq[0]),)+ map(fn,seq[1:])`	

Example of a predicate	`is_odd = lambda x : x%2 != 2`	
Example of accumulate fn	`add = lambda x, y: x + y`	Binary operation, takes in 2 i/ps
Enumerate -> Filter -> Map -> Accumulate		

STRING / TUPLE		
String slicing	`s[start:stop:step]`	`stop`: not inclusive `step = 2`: to skip an element `step = (-)ve`: steps taken in decreasing index
Tuple index slice		Valid range for tuple: -n <= x <= n - len(tuple) = n - (- n) for negative index slice
Obtain start position of string	`my_string.find('dog')` `S1.find(S2, position)`	Find string S2 in S1 starting from index position
	`seq.index(num)`	Only gives the index of first num it finds in the string/tuple
Replace string	`my_string.replace('dog','cat')`	Replace all occurrences of 'dog' in my_string to 'cat'
Reverse tuple	`tuple(reversed(tuple))`	

Finding max or min	`max(a, b)`
	`max(a, b, c…)`
	`min(a, b)`	

	`element in x`	Returns True if element in x, False otherwise
	`for i in x`	
More than, less than	`str_1 > str_2` `tpl_1 < tpl_2`	Base on: 1) Alphabetical 2) # of letters/element
Split	`Tuple(line.split(','))`	Splits into list

Mathematical Identities:

How to write recursion:

1. Figure out the base case
   - Typically n = 0 or n = 1
2. Assume you know how to solve n-1
   - Now how to solve for n?

Order of Growth:

1. Time
2. Space

- Count the number of "basic computational steps"
  - ❖ Identify the basic computation steps
  - ❖ Try for a few small values of n
  - ❖ Look for "worst case" scenario

Complicated Algorithms:

Print out each character one at a time	`index = 0`
	`while index < len(fruit):`
	`  letter = fruit[index]`
	`  print(letter)`
	`  index += 1`

Taking last digit in a number	`num%10`

Finding prime num	`from math import *`
	
	`def is_divisible(x, i):`
	
	`def is_prime(x):`
	`  if x == 1:`
	`  return False`
	`  else:`
	`  for i in range(2, ceil(sqrt(x))):`
	`  if is_divisible(x, i):`
	`  return False`
	`  return True`

Count the # of occurrences (letter)	`def count_letter(string):`
	`  counter, index = 0, 0`
	`  for letter in string:`
	`  if letter == 'A':`
	`  counter += 1`
	`  return counter`

Count the # of occurrences (substring)
 	
- Counts substrings that begin with 'A' and end with 'X'
-E.g. CAXAAYXZA
- index = # of As
- counter: if X, add all the # of As before it
	`def count_substring(string):`
	`  counter, index = 0, 0`
	`  for letter in string:`
	`  if letter == 'A':`
	`  index += 1`
	`  if letter == 'X':`
	`  counter += index`
	`  return counter`

Count the number of occurrences of S2 in S1 (no overlap)
 	
E.g. 'CS1010S' has 2 occurrences of '10'
E.g. '110101' has only 1 occurrence of '101'
	`def occurrence(S1, S2):`
	`  counter, position = 0, 0`
	`  length = len(S2)`
	`  while position < len(S1):`
	`  position = S1.find(S2, position)`
	`  if position == -1:`
	`  break`
	`  counter += 1`
	`  return counter`

Fast Exponential
(b^e)
 	
Time: O(log n)
Space: O(log n)
	`def fast_expt(b,e):`
	`  if e == 0:`
	`  return 1`
	`  elif e%2 == 0:`
	`  return fast_expt(b*b, e/2)`
	`  else:`
	`  return b * fast_expt(b, e-1)`

TOWER OF HANOI

1. Move n-1 discs from A to B using C
2. Move disc from A to C
3. Move n-1 discs from B to C using A

```
def move_tower(size,src,dest,aux):
    if size == 1:
        print_move(src, dest)
        #display the move
    else:
        move_tower(size-1,src,aux,dest)
        print_move(src, dest)
        move_tower(size-1, aux, dest,src)

def print_move(src, dest):
       print("move top disk from", src,
       "to", dest)
```

```
def hanoi(n, src, dsc, aux):
    if n == 0:
       return ()
    elif n == 1:
       return ( (src,dsc) ,)
    else:
       return hanoi(n-1, src, aux, dsc)+
              ((src,dsc),)+
     hanoi(n-1,aux,dsc,src)
```

COUNT CHANGE

1: 1 cent
2: 5 cent
3: 10 cent
4: 20 cent
5: 50 cent
6: 100 cent

Time: leaves in tree
: $O(2^{a+d})$
: $O(2^{a})$
Space: depth of entire tree
: O(a)

Where a = amount,
d = denomination (fixed)

```
def cc(amount, kinds_of_coins):
    if amount == 0:
        return 1
    elif (amount<0) or (kinds_of_coins<0):
        return 0
    else:
        return
        cc(amount -
     first_denomination(kinds_of_coins)),
        kinds_of_coins)+
        cc(amount, kinds_of_coins -1)

def first_denomination(kinds_of_coins):
    if kinds_of_coins <= 0:
        return None
    elif kinds_of_coins == 1:
        return 1
    elif kinds_of_coins == 2:
        return 5
                ...
```

Counting leaves

```
def count_leaves(tree):
    if tree == ():
        return 0
    elif is_leaf(tree):
        return 1
    else:
        return count_leaves(tree[0])+
          count_leaves(tree[1:])
```

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

Flatten the tree

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