def makettable(s1,s2):

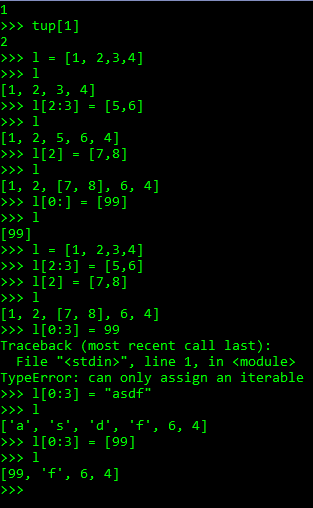


table = {}

for i in range(len(s1)):

table[s1[i]] = s2[i]

return table

def translate(ttable, s):

t = ""

for c in s:

t += ttable.get(c, c)

return t

def sort\_same\_freq(l, rev):

returnList = []

listDi = []

for (key, value) in l:

if len(listDi) > 0:

# Sort if we reached a new number of occurances

if value != listDi[0][1]:

# attach to return list

returnList.extend(sorted(listDi, reverse=rev))

listDi = []

# Add tuple

listDi.append((key,value))

# Add last sorted list

returnList.extend(sorted(listDi, reverse=rev))

return returnList

def histo(s):

d = {}

# Make histogram

for c in s:

# Don't need newline or return line

if (c != '\n') and (c != '\r'):

d[c] = d.get(c, 0) + 1

# Return sorted table

return sort\_same\_freq(sorted(d.items(), key=lambda item: (item[1], item[0])), True)

def digraphs(s):

l = {}

# Create pair and add to dictionary

for i in range(len(s)-1):

if (s[i] != '\n') and (s[i] != '\r'):

f = '/'+s[i]+s[i+1]+'/'

l[f] = l.get(f, 0) + 1

# sort by same value; sort same freq in increasing order

return sort\_same\_freq(sorted(l.items(), key=lambda item: (item[1], item[0]), reverse=True), False)

//returns: new concatenated list of list1 + list2

(append '(1 2 3) '(4 5 6))

(1 2 3 4 5 6)

(define (myappend l1 l2)

(cond

((null? l1) l2)

(#t (cons (car l1) (myappend (cdr l1) l2)))

))

(define (badreverse l1)

(cond

((null? l1) '())

(#t (append (reverse (cddr l1)) (list (car l1))))

))

(define (revappend l1 l2)

(cond

((null? l1) l2)

(#t (revappend (cdr l1) (cons (car l1) l2)))

))

(define (fold combinef base l1)

(cond

((null? l1) base)

(#t (combinef (car l1) (fold combinef (cdr(l1)))))

))

(define (addup l1) (fold + 0 l1))