```
str1 = "L DVEE" Str120:1] ="LOVE" Str12:-1] ="LOVE"
                                                                  Postscript Notes
                                         everything but the
                                                                  code blocks are evaluated when they are called
           01234
                                                                 Pro: no parenthesis or precedence needed
          -5 -4 -3 -2-
   *orrings are immutable => can't change part of it, can reass.
                                                                 index so copies that index (of from top) to the top
   STY I ="LOVEE" &> STY[I]=X"BAD &) STY = "LOVXE" GOOD
                                                                 roll => positive: from top down, rug: botton up
   todid soon hower totist of diff types 51 = " Lovee - Baccus"
                                                                 def = appends to the end of the top dict
   51. split ("-", 1) = ["Lover", "Bacous"] split (delin, # of splits)
                                                                 Dict = pushes empty dict to the op stack
                                                                 Begin = takes dict from Op Stack 3 pushes to Dict Stack
   List's
   * Lists are mutable => LI = [1,2,3] x) LI[1:3] = 4 => [0,4]
                                                                 End = popo top dict from dict stack, discards
   * Litts were append => LI. append (91)
   12=["a","b","c"]"-".join(L2)=9"a-b-c"
                                                                 1x 3 def
                                                                                            (91)
                                                                 1913 /x 9 def x3 def
   WANTEDON'S STORESTORE
   sorted (iterable, key, reverse) => returns a new list
                                                                                            d
                                                                 f1
    can't sort a light w/ multiple types in it
    can sort list of list key can set a diff element to
                                                                 x mul
                                                                                                     EM: >
  LI. sort () => changes the og list
                                                                                                      £1:3-33
                                                                                                      Dict
  Tuples & Tuples are iterable "
  [1:x] notation is fine * Tuples are immutable, can reassign
  Extract Patterns => x, y, 2 = (1, 2, 3) * works for strings too
                                                                 1x 3 def
  can caste between List (m) on Tuple (m)
                                                                                        def x end 3 def
                                                                /fl 31 dict begin
 Anon Functions lambda x: X+1
Dictionaries keys must be Hashable of immutable
                                                                                        end
if the key DNE, trying to use dix's notation is Bad at use digeth
                                                                x mul
d.get (5,-1) looks for 5, but returns -lif 5 DNE
d[5]=[1,2,3] will add (5:[1,2,3]) to the diet
list (d. keys ()/d. values ()/d. items) d. items returns a list
                                                                                         dict
Sorted (d) => returns sorted list of the leays
Sorbed (list (d. items (s)) => returns list sorted by the Kay
to Den't forget to cast back to a Dict, if desired
                                                                                          def
Sets no duplicates -> Set I = set (List I) // removes dups
                                                                                                    Dict
                                                                                     OP
log = {'Mar':2706,'Apr':3620,'May':1860,'Jun':2157,'July':5014,'Aug':4327,'Sep':2843}
from functools import reduce
sum log = lambda log: reduce(lambda x,y: x+y, log.values())
```

alltuples = [('Mar', 4614), ('Apr', 5959), ('May', 3152), ('Jun', 4185), ('July', 11487), ('Aug', 9281), ('Sep', 7403), ('Oct', 278)]

tlog = ('King', {'Mar':2706,'Apr':3620,'May':1860,'Jun':2157,'July':5014,'Aug':4327,'Sep':2843})

('King', 22527)

#3

helper = (lambda tlog: (tlog[0], sum\_log(tlog[1])))

helper = lambda x,y : x if x[1] > y[1] else y

helper(('Mar', 4614), ('Apr', 5959))

reduce(helper, alltuples)

map( lambda tlog: (tlog[0], sum log(tlog[1])), monthlyCases.items() )

```
## problem 4a) searchDicts(L.k)
def searchDicts(L,k):
    for d in reversed(L):
       dKey = list(d.keys())
        #check if k is one of the keys in the dictionary
        if (dKey[0] == k or d.get(k)):
           return d.get(k)
## problem 4b) searchDicts2(L,k)
def searchDicts2 helper(tL,k,ind):
    if k in tL[ind][1]:
        return tL[ind][1][k]
    else:
        if ind == 0:
           return None
            next ind = tL[ind][0]
            return searchDicts2_helper(tL,k,next_ind)
```

## 3) [20 pts]

(a) [8 pts] Define a Python function, aroundL, that takes a list of integers as input and returns a list of pairs containing one more and one less than each number in the list. For example, aroundL ([1,2,3]) should return [(0,2),(1,3),(2,4)] as its answer. Your function may involve a loop or can be recursive.

```
def aroundL(L):
    result = []
    for item in L:
       result.append((item-1, item+1))
    return result
```

(b) [6 pts] Re-write the aroundL function from problem 3(a) using list comprehension.

```
def aroundL(L):
    return [(item-1, item+1) for item in L ]
```

(c) [6 pts] Re-write the  $\mathtt{aroundL}$  function from problem 3(a) using high order functions (map, reduce, or filter).

```
def aroundL(L):
    return list(map(lambda x: (x-1,x+1),L))
```

## 4) [16pts]

(a) [10 pts] Define a Python function combine that takes two dictionaries as argument and combines them by merging values of the common keys as lists. The output dictionary includes only the common keys from the two input dictionaries. The order of the elements in the output dictionary can be arbitrary.

Your solution may use loops.

```
For example:
```

## (b) [10 pts]

Re-write the combine function from problem 4(a) using high order functions (map, reduce, or filter).

```
def combine(d1,d2):
    filtered = filter(lambda t: t[0] in d2, d1.items())
    return dict(map(lambda x: ( x[0] , [ x[1], d2[x[0]] ]),filtered))
```

1) [15 pts] Directions: Consider the following Python code.

```
def add(a):
    global x
    x = a+x
    return x

def grow (c):
    return x+c

def outer (f):
    def inner (g, y):
        return f(y)+g(y)
    return inner
```

For each of the following Python expressions, write the valuthat the expressions are evaluated in a new session (i.e., ans previous lines)

```
a) x=1
  L=[1,2,3,4,5]
  list(map(grow,L))

[2, 3, 4, 5, 6]

b) x=1
  L=[1,2,3,4,5]
  list(map(add,L))

[2, 4, 7, 11, 16]

c) x=1
  outer(add)(grow,5)

17

d) x=2
  outer(add)(grow,x)
```

outer(grow)(add,x)

e) x=2

8