```python
# Generates a sequence of natural numbers from init to N (inclusive)
class Naturals(object):
    def __init__(self,init):
        self.current = init
        self.N = N
    def __next__(self):
        if self.current > self.N:
            raise StopIteration
        result = self.current
        self.current += 1
        return result
    def __iter__(self):
        return self
```

```python
# Global or local variable?
s = "CptS355"
def f():
    global s # allows you to access the variable s within the function
    print(s)
    s = "CptS322" # changes the value of s

f() # prints CptS355
print(s) # prints CptS322
```

```python
def histo2(s):
    l = list(set([(c, s.count(c)) for c in s]))
    return sorted(sorted(list(l)), key = lambda item : item[1], reverse=True)

histo2("implemented") # [('e', 3), ('m', 2), ('d', 1), ('i', 1), ('l', 1), ('n', 1), ('p', 1), ('t', 1)]
```

```python
import copy
from functools import reduce
def sumSalesN(L):
    def combine_dicts_ho(d1,d2):
        d = copy.deepcopy(d1)
        __common_items = map(lambda x: (x[0],x[1] + d2.get(x[0],0) ), d.items())
        __other_items = filter(lambda x: x[0] not in d1.keys(), d2.items())
        return dict(list(__common_items) + list(__other_items))
    return dict(sorted(list(reduce(combine_dicts_ho, list(map(sumSales,L))).items())))

allSales = [{'Amazon':{'Mon':30,'Wed':100,'Sat':200},
             'Etsy':{'Mon':50,'Tue':20,'Wed':25,'Fri':30},
             'Ebay':{'Tue':60,'Wed':100,'Thu':30},
             'Shopify':{'Tue':100,'Thu':50,'Sat':20}},
            {'Shopify':{'Mon':25},
             'Etsy':{'Thu':40, 'Fri':50},
             'Ebay':{'Mon':100,'Sat':30}},
            {'Amazon':{'Sun':88},
             'Etsy':{'Fri':55},
             'Ebay':{'Mon':40},
             'Shopify':{'Sat':35}} ]
sumSalesN(allSales) # {'Fri': 135, 'Mon': 245, 'Sat': 285, 'Sun': 88, 'Thu': 120, 'Tue': 180, 'Wed': 225}
```

```python
def getCourse(it):
    course = ""
    for c in it:
        if c == '-':
            return course
        else:
            course += c
    return course

i = iter("CptS355-CptS322-CptS321")
course1 = getCourse(i)
print(course1) # CptS355
course2 = getCourse(i)
print(course2) # CptS322
course3 = getCourse(i)
print(course3) # CptS321
```

```python
# Generate an iterator which takes an iterator of strings and generates a sequence of strings
# where each pair of consecutive strings from the input iterator are concatenated.
class concat_consecutive(object):
    def __init__(self,it):
        self.input = it
        self.current = self.get_next()
    def get_next(self):
        try:
            current = self.input.__next__()
        except:
            current = None
        return current
    def __next__(self):
        if self.current is None:
            raise StopIteration
        n = 2
        word = ""
        while n>0:
            word += self.current
            self.current = self.get_next()
            if self.current is None:
                return word
            n -= 1
        return word
    def __iter__(self):
        return self
it = concat_consecutive(iter(["CptS","355","CptS","322","CptS","321","done"]))
for s in it:
    print("Next string: ", s)
#Next string:  CptS355
#Next string:  CptS322
#Next string:  CptS321
#Next string:  done
```

```python
def sumSales(sales):
    d = {}
    for store,log in sales.items():
        for day,sale in log.items():
            d[day] = d.get(day,0) + sale
    return dict(sorted(list(d.items())))

mySales = {'Amazon':{'Mon':30,'Wed':100,'Sat':200},
           'Etsy':{'Mon':50,'Tue':20,'Wed':25,'Fri':30},
           'Ebay':{'Tue':60,'Wed':100,'Thu':30},
           'Shopify':{'Tue':100,'Thu':50,'Sat':20}
          }
sumSales(mySales) # {'Fri': 30, 'Mon': 80, 'Sat': 220, 'Thu': 80, 'Tue': 180, 'Wed': 225}
```

```python
# Creates a copy of the input iterable object
class copyIter(object):
    def __init__(self,it):
        self.input1 = it
        self.current = self._getNextInput()
    def _getNextInput(self):
        try:
            current = self.input1.__next__()
        except:
            current = None
        return current
    def __next__(self):
        if self.current is None:
            raise StopIteration
        result = self.current
        self.current = self._getNextInput()
        return result # can return the result before the stopIteration is called
    def __iter__(self):
        return self

it = copyIter(iter("ABCDEFG"))
for c in it:
    print(c) # A B C D E F G
```

```python
def histo(s):
    d = {}
    for c in s:
        d[c] = d.get(c,0) + 1 # if c is not a key, 0 will be returned. If c is a key, then the value will be returned
    return sorted(sorted(list(d.items())), key = lambda item : item[1], reverse=True) #sort by [1] unless there is a tie, then sort by [0]

histo("implemented") # [('e', 3), ('m', 2), ('d', 1), ('i', 1), ('l', 1), ('n', 1), ('p', 1), ('t', 1)]
```

(a) 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:
```
combine( {1:'A',2:'B',3:'C',4:'D',5:'E',6:'F'},
         {5:50,1:10,6:60,2:20,7:78,9:90} )
```
returns
```
{1: ['A', 10], 2: ['B', 20], 5: ['E', 50], 6: ['F', 60]}
```

```
def combine(d1, d2):
    d = {}
    for k,v in d2:
        if k in d2:
            d[k] = [d1.get(k), d2.get(k)]
    return d
```

(b) Re-write the combine function 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))
```

```
L = [1, 2, 3]
S = (1, 2, 3)
C = '1 2 3'
D = {3:4, 2:5, 1:6}
```

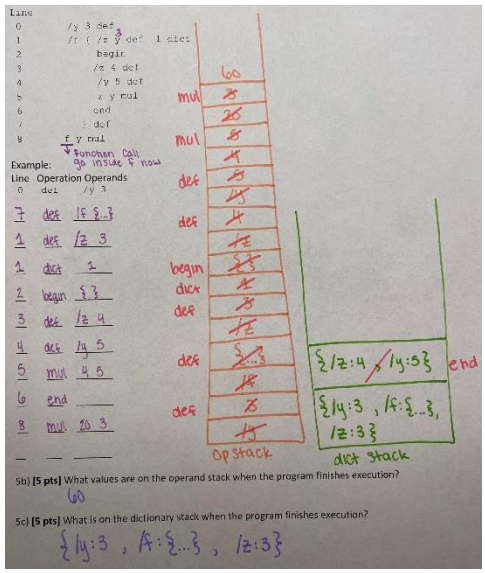
a) What is L[-1]?
3

b) What is C[1] (make sure you give an answer of the right type!)?
' ' #assume there is a space between numbers

c) What is S[1:-1]?
(2,) # need the comma to indicate it is a tuple

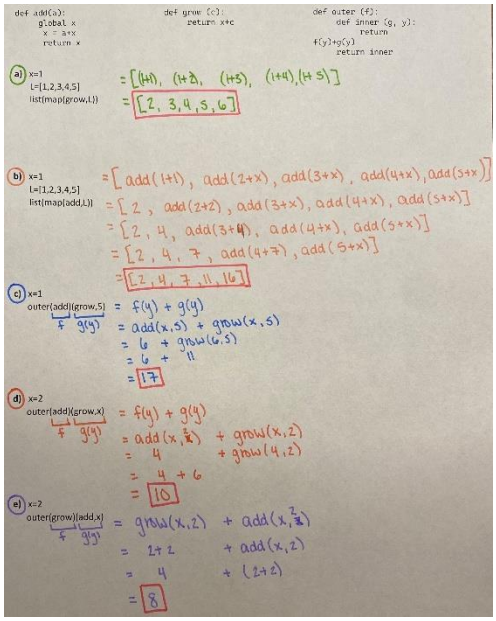
d) What is the value of C after executing C[1] = '3'?
Will give an error because strings are immutable in Python.

e) What is the value of D[3]?
4



5b) **[5 pts]** What values are on the operand stack when the program finishes execution?
60

5c) **[5 pts]** What is on the dictionary stack when the program finishes execution?
{ly:3 , lf:{...} , lz:3}



(a) 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) Re-write the aroundL function using list comprehension.

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

(c) Re-write the aroundL function using high order functions (map, reduce, or filter).

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

```
class mystery_iter():
    def __init__(self,iIterable):
        self.iterator = iIterable
        self.L = []

    def __next__(self):
        result = None
        for x in self.iterator:
            if x not in self.L:
                self.L.append(x)
                result = x
                break

        if result == None:
            raise StopIteration
        return result

    def __iter__(self):
        return self
```

(a) Explain what sequence the mystery_iter iterator represents.
The unique values from the Herable input value.

(b) Evaluate the following expressions and write down the output of the print statements.

```
myI = mystery_iter(iter([1,2,1,3,2,1,5,5]))
myI.__next__() #skips 1
for i in myI:
    print(i)
```
self.L = [1, 2, 3, 5]

2
3
5

```
myI = mystery_iter(iter(['a','r','s','a','r', 3, 5, 5]))
for i in myI:
    print(i)
```
'a'
'r'
's'
3
5

```
# local vs global variables
def demo():
    L[0] = 'c'
    return L

L = [1,2]
result = demo() # ['c',2]

#--------------
def demo(L):
    L[0] = 'c'
    return L

L = [1,2]
result = demo(L) # ['c',2]
```

```
# local vs global variables
def demo2():
    L=['a','b','c']
    return L

L = [1,2]
result = demo2()
print(result) # ['a', 'b', 'c']
print(L) # [1,2]

#-------------
def demo2(L):
    L=['a','b','c']
    return L

L = [1,2]
result = demo2(L)
print(result) # ['a', 'b', 'c']
print(L) # [1,2]
```

List Comprehension example

Loop method:
```
L = []
for x in range(1,4)
L.append(x+x)
```

List Comp. method:
```
[x+x for x in range(1,4)]
```

```
# global, local, and nonlocal variables
z = 1
def f():
    z = 4
    def g(a):
        nonlocal z # refers to z = 4 (not the global one but the one right outside of its scope)
        print("z in g:",z) # z = 4
        z =  10
        return z # z = 10
    return g(19) + z # 10 + 10 =20

result3 = f()
print("example 3 - z in main:",z) # z = 1
```

```
# global and local variables
z =1
def f():
    z = 4
    def g(a):
        global z
        print("z in g:",z)
        z = 10 # refers to the global z defined outside of the function
        return z
    g(10)
    print("Inside f:",z)
    return z+g(19) # 4 + 10 = 14

result2 = f()
print("example 2 - z in main:",z)

# Output:
#    z in g: 1
#    Inside f: 4
#    z in g: 10
#    example 2 - z in main: 10
```

```
# sorting list of tuples
myL = [('a',3), ('b',2), ('f',1),('d',1), ('e',1), ('c',1)]
sorted(myL) # [('a', 3), ('b', 2), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
sorted(myL, key = lambda item: item[1]) # [('f', 1), ('d', 1), ('e', 1), ('c', 1), ('b', 2), ('a', 3)]
# sort first on the second value then the first
sorted(sorted(myL), key = lambda item: item[1]) # [('c', 1), ('d', 1), ('e', 1), ('f', 1), ('b', 2), ('a', 3)]
```