1. What is printed by each of these programs? If the program contains an error, explain it.

(a)

```
x = [1,2,3]

def f1(z):
    z[0] = z[2] + z[2]
    return z
    print ’z returned!’

print f1(x)
print x
```

Solution:

```
[6, 2, 3]
[6, 2, 3]
```

(b)

```
def f2(x):
    y = x
    x = {0:’a’, 1:’b’, 2:’c’}
    y[−1] = ’e’

x = [’a’,’b’,’c’]
print f2(x)
print x
```

Solution:

```
None
[’a’, ’b’, ’e’]
```

(c)

```
def f3(x):
    y = {}
    for k in x:
        y[x[k]] = k
    return y

x = {’sun’:’good’, ’rain’:’bad’}
f3(x)
print y.keys()
```

Solution:

```
Traceback (most recent call last):
  File "exercise_f3.py", line 9, in <module>
    print y.keys()
NameError: name ’y’ is not defined
```

(d)
```
def f4(x,y):
    t = x
    x = y
    y = t
    x[’sun’] = 2
    y[’rain’] = 0

a = {’sun’:0}
b = {’rain’:2}
f4(a,b)
print a
print b
```

Solution:

```
{’sun’: 0, ’rain’: 0}
{’sun’: 2, ’rain’: 2}
```

2. Write a *recursive* function `has_6` that takes a list of numbers and returns `True` if the list contains a 6 and `False` otherwise. You **cannot** use the `in` operator, loops, etc.

Solution:

```
def has_6(L):
    '''(list of int) -> bool
    Returns True if 6 is in L, False otherwise.
    >>> has_6([1,2,6,3])
    True
    >>> has_6([1,2,3])
    False
    '''
    if len(L) == 0:
        return False
    elif L[0] == 6:
        return True
    else:
        return has_6(L[1:])
```

3. Write a *recursive* function `index_6` that takes a list of numbers and returns the index of 6 in the list or -1 if it's not in the list. You **cannot** use the `index` method, loops, etc.

   Hint: consider the following snippet of code. What does `position` equal? It's not 2!

```
L = [8, 7, 6, 13]
position = index_6(L[1:])
```

Solution:

```
def index_6(L):
    '''(list of int) -> int
    Returns the index of 6 in L, -1 if 6 not in L.
    >>> index_6([1,2,6,3])
    2
    >>> index_6([1,2,3])
    -1
    '''
    if len(L) == 0:
        return -1
    elif L[0] == 6:
        return 0
    else:
        idx = index_6(L[1:])
        if idx == -1:
                return -1
        else:
                return idx+1
```

4. We have a number of bunnies and each bunny has two big floppy ears. We want to compute the total number of ears across all the bunnies recursively. Write a *recursive* function `bunny_ears` that takes in a number and returns the number of ears. You **cannot** use loops or multiplication.

   Examples:

```
>>> bunny_ears(0)
0
>>> bunny_ears(1)
2
>>> bunny_ears(2)
4
```

Solution:

```
def bunny_ears(how_many):
    '''(int) -> int
    Returns the number of ears for how_many bunnies.
    >>> bunny_ears(0)
    0
    >>> bunny_ears(1)
    2
    >>> bunny_ears(2)
    4
    '''
    if how_many == 0:
        return 0
    else:
        return 2 + bunny_ears(how_many-1)
```

5. Write a *recursive* function `count_hi` that takes a string and returns the number of times lowercase `'hi'` appears in the string. You **cannot** use the `find` method, loops, etc.

Solution:

```
def count_hi(s):
    '''(str) -> int
    Returns the number of times 'hi' occurs in s.
    >>> count_hi('xxhixx')
```

```
1
>>> count_hi('hixxhixx')
2
>>> count_hi('hxixhxix')
0
>>> count_hi('xhi')
1
'''
if len(s) <= 1:
    return 0
elif s[:2] == 'hi':
    return 1 + count_hi(s[2:]) # safe to skip 2
else:
    return count_hi(s[1:]) # only move 1 (see last example in docstring)
```

Bunnies and counting “hi” problems adapted from Nick Parlante (Codingbat).