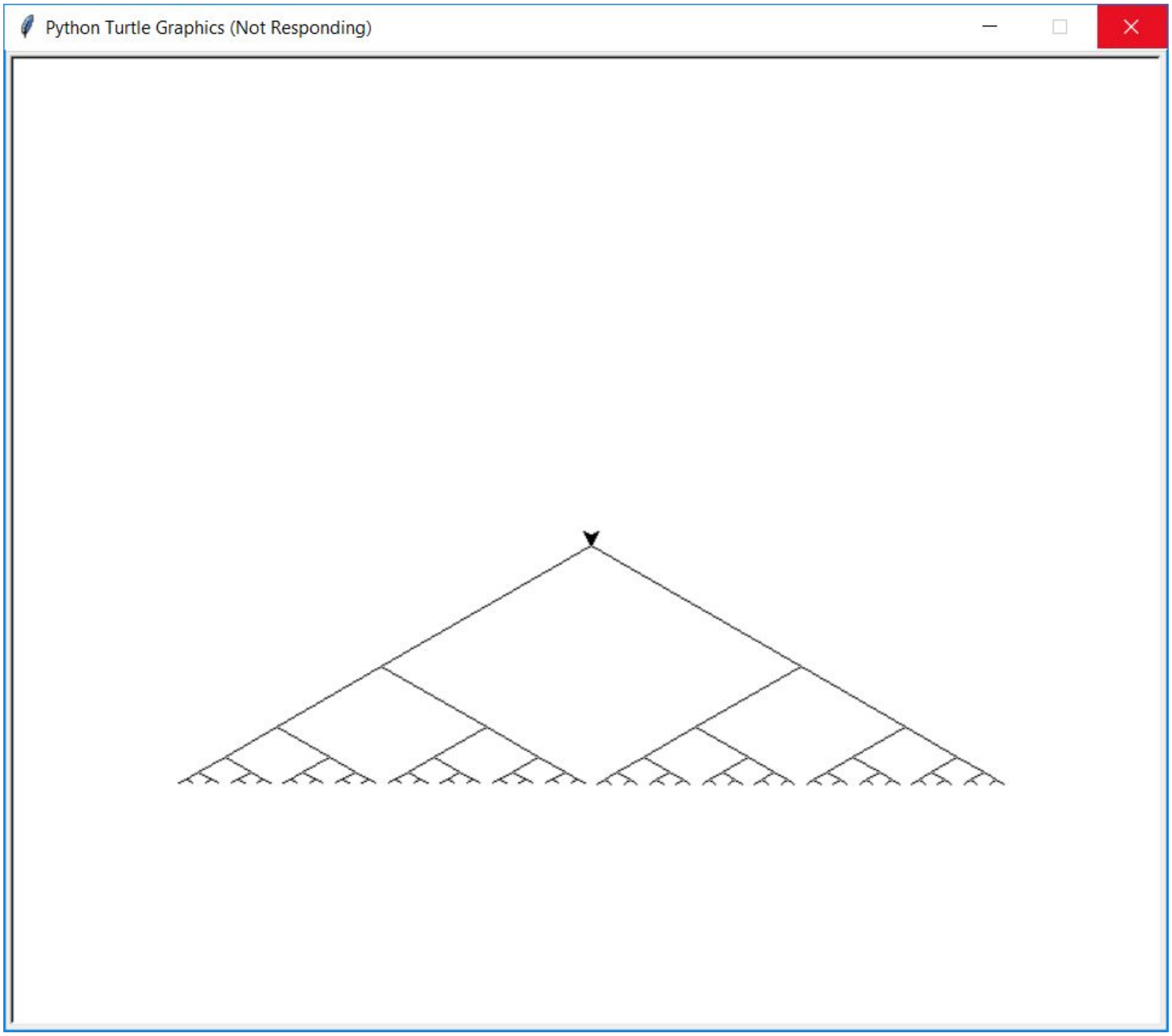
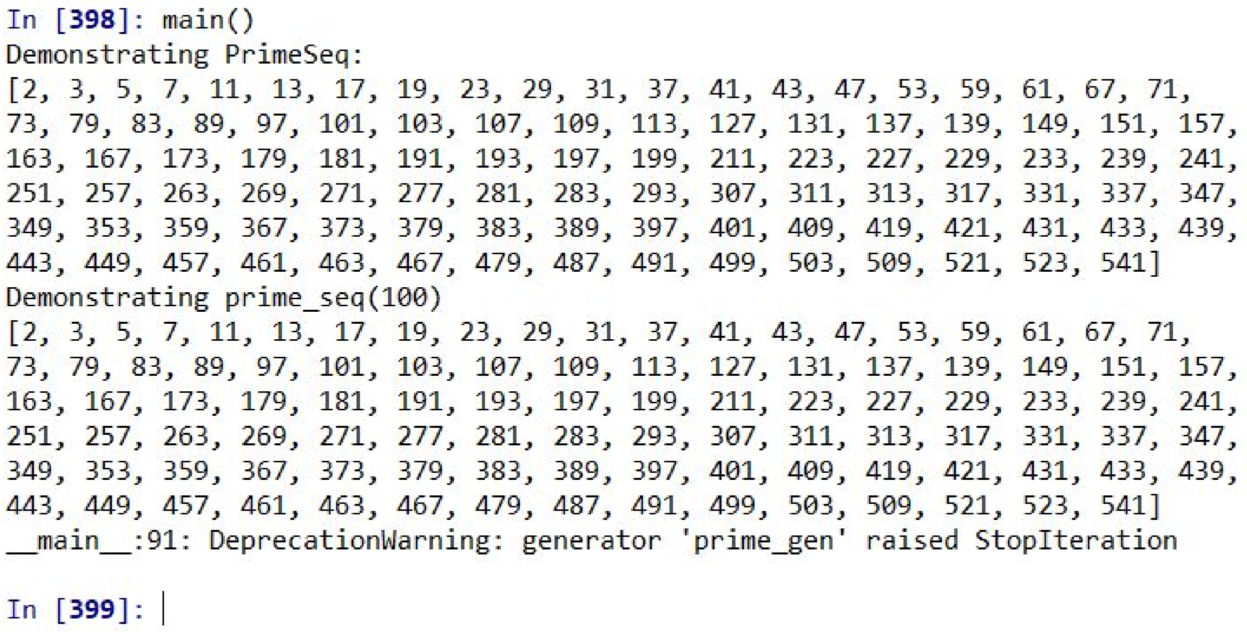
Problem 1

```
#Michael Quach
from turtle import *
import turtle
result = {}
#Problem 1. Recursive Functions
#a) Binary Tree.
def binary_tree(depth, length):
    '''Recursively draws a binary tree. Assumes that the turtle is pointing down on start.
    Returns the turtle to its starting position, but the board will obviously have the tree drawn on
it.'''
    if(depth<=0):
        return
    #Drawing left (of turtle) branch
    left(60)
    forward(length)
    right(60)
    #Drawing branches of the left branch
    binary_tree(depth-1, length/2)
    #Going back to the starting point
    left(60)
    backward(length)
    #Drawing right (of turtle) branch
    right(120)
    forward(length)
    left(60)
    #Drawing branches of the right branch
    binary_tree(depth-1, length/2)
    #Going back to the starting point
    right(60)
    backward(length)
    left(60)
    #Done()

#Part
b:############################################################################
def power(x,n):
    '''Returns the value of x raised to the n power, computed with recursive
    divide-and-conquer. Equivalent to x**n. x and n must be numbers or support
    several standard numerical operators (e.g. '<=', '*', '%').'''
    if(n<=0):
        return 1
    elif(n%2 != 0):
        return x*power(x,(n-1)//2)*power(x,n//2)
    else:
        return power(x, n//2) * power(x, n//2)

def test_power():
    '''Tests power(). The tests object is initialized with a sample of pairs of values
    and then iterated through to test each value pair in power(). Uses the testif()
    function, which prints to the console. Result is compared to using the ** operator.'''
    #Initialize test object with a bunch of tests
    tests=[]
    for ii in range(10):
        for jj in range(10):
            tests.append((ii, jj))
    print("Test names are in format: Function ({x},{y})")
    for ii in range(len(tests)):
        testname = "power "+str(tests[ii])
        testif(power(tests[ii][0], tests[ii][1]) == tests[ii][0]**tests[ii][1], testname)

#Part c)  Memoized Slice Sums.#####################################################
def slice_sum(lst, begin, end):
    '''Returns the sum of elements in lst, starting with position begin and ending at (excluding)
end.
    begin and end parameters must be positive and less than the length of lst, or else an
IndexError is raised.
    Result is compared to expression sum(lst[begin:end]).'''
    if(begin < 0 or begin > len(lst) or end < 0 or end > len(lst)):
        raise IndexError
    elif(begin >= end):
        return 0
    else:
        return lst[begin] + slice_sum(lst, begin+1, end)

def test_slice_sum():
    '''Tests slice_sum(). The tests object may be edited to contain any summable collections.
    This function attempts to iterate through every possible permutation of the collections given
using testif().
    Note that testif() prints to console, so this function may clutter it.'''
    tests = [[1,2,3,4,5,6,7,8,9],
             [9,8,7,6,5,4,3,2,1],
             [4,87,8,2,5,4,8,2,5,100],
             [486,125,753,951,0,500,2,56,985,653,111,325,0,9]]
    begin, end = 0,0
    print("Test names are in format: Function {ii}.{begin}.{end}")
    for ii in range(len(tests)):
        for begin in range(len(tests[ii])):
            for end in range(len(tests[ii])):
                testname = "slice_sum "+str(ii)+"."+str(begin)+"."+str(end)
                testif(slice_sum(tests[ii], begin, end) == sum(tests[ii][begin:end]), testname)

def slice_sum_m(lst, begin, end):
    global result
    if(begin < 0 or begin > len(lst) or end < 0 or end > len(lst)):
        raise IndexError
    if(begin == end):
        return 0
    else:
        result[(begin, end)] = lst[begin] + slice_sum_m(lst, begin+1, end)
        return result[(begin, end)]
    return result

def test_slice_sum_m():
    '''Tests slice_sum_m(). The tests object may be edited to contain any summable collections.
    This function attempts to iterate through every possible permutation of the collections given
using testif(),
    which prints to console.'''
    tests = [[1,2,3,4,5,6,7,8,9],
             [9,8,7,6,5,4,3,2,1],
             [4,87,8,2,5,4,8,2,5,100],
             [486,125,753,951,0,500,2,56,985,653,111,325,0,9]]
    begin, end = 0,0
    for ii in range(len(tests)):
        for begin in range(len(tests[ii])):
            for end in range(len(tests[ii])):
                testif(slice_sum_m(tests[ii], begin, end) == sum(tests[ii][begin:end]))

def testif(b, testname='', msgOK='', msgFailed=''):
    '''Function used for testing.
    param b: boolean, normally a tested condition: true if passed, false otherwise
    param testname: the test name
    param msgOK: string to be printed if param b==True (test condition true
    param msgFailed: string to be printed if param b==False (test condition flae)
    returns b'''
    if b:
        print("Success:", str(testname), ";", msgOK)
    else:
        print("Failed:", str(testname), ";", msgFailed)
    return b
```



Problem 2

```
#Michael Quach
#Problem 2. Prime Number Generator
#Part a). Iterator Class
class PrimeSeq:
    '''Class that generates the first n prime numbers in sequence.
    n should be greater than 0.'''
    def __init__(self, n):
        self.n = n
        self.x = 2
        self.__primes = []
    def __iter__(self):
        return self

    def __next__(self):
        while(self.n>=0):
            if(self.n<=0):
                raise StopIteration
            if(len(self.__primes) == 0):
                self.__primes.append(self.x)
                self.n-=1
                self.x+=1
                return self.x-1
            else:
                primed=True
                for ii in self.__primes:
                    if((self.x/ii).is_integer()):
                        self.x+=1
                        primed=False
                        continue
                if(primed):
                    self.__primes.append(self.x)
                    self.n-=1
                    self.x+=1
                    return self.x-1

#Part b). Generator
def prime_gen(n):
    '''Generator for a sequence of the first n prime numbers.
    n should be greater than 0, else StopIteration is raised. No side effects.'''
    x=2
    while n>=0:
        if(n<=0):
            raise StopIteration
        if(x==2):
            n-=1
            yield x
            x+=1
        else:
            primed=True
            for ii in range(2,x):
                if((x/ii).is_integer()):
                    x+=1
                    primed=False
            if(primed):
                n-=1
                yield x
                x+=1

def main():
    '''Demonstrates PrimeSeq class and prime_seq function.'''
    print("Demonstrating PrimeSeq:")
    seq1 = PrimeSeq(100)
    primes1 = [x for x in seq1]
    print(primes1)

    print("Demonstrating prime_seq(100)")
    seq2 = prime_gen(100)
    primes2 = [x for x in seq2]
    print(primes2)
```



```
In [398]: main()
Demonstrating PrimeSeq:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157,
163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347,
349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541]
Demonstrating prime_seq(100)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157,
163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347,
349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541]
__main__:91: DeprecationWarning: generator 'prime_gen' raised StopIteration

In [399]: 
```

```
#Michael Quach
#Problem 3. Functional Code with Random Number Sequences
import random
import itertools
def gen_rndtup(n):
    '''Generates a random two-tuple where each element is less than n.'''
    a,b=0,0
    while(n):
        a, b = random.randint(1,n), random.randint(1,n)
        yield (a,b)
#Write a generator gen_rndtup(n) that creates an infinite sequence of tuples
#(a, b) where a and b are random integers, with 0 < a,b < n. If n == 7, then
#a and b could be the numbers on a pair of dice. Use the random module.
#lambda
#islice()
#filter
def partA():
    n=7
    itera = itertools.islice(iter(filter(lambda a,b,n: a+b >= n//2, gen_rndtup(n))), 10)
    for ii in range(10):
        print(next(itera))
#a) Write code in file p3.py that uses lambda expressions, the itertools.islice
#function (https://docs.python.org/3/library/itertools.html#itertools.islice),
#and the filter function to display the first 10 generated tuples (a, b) from
#gen_rndtup(7) that have a + b >= n // 2.
#Example: with n==7 the output could be: (4,1), (2,6), (6,6),(3,5),...

def generatorB():
    a,b=0,0
    while True:
        a,b = random.randint(1,14), random.randint(1,14)
        yield (a,b)

def filterB(num, seq):
    n=7
    while num:
        a,b = next(seq)
        if(a+b >= n//2):
            num-=1
            yield (a,b)

def partB(num):
    seq = (seq for i in filterB(num, generatorB()))
    for ii in seq:
        print(next(seq))
#b) Write code using generator expressions and one for loop that displays the
#first 10 random integer tuples (a, b), with 0<a,b<n, where a + b >= n // 2
#and n being a positive integer local variable initialized with value 7.
#Do not use the gen_rndtup(n) generator from part a). You may use other functions.
#Place all the code in file p3.py and paste that in h6.doc.
#
```

```
#Extra credit part, for 5 points:
#c) Use lambda expressions, map(), the itertools.islice, functools.reduce(),
#and the filter function to display the sum of first 10 generated tuples
#(a, b) that have sum a + b >= n // 2.
#The sum of tuples is done component-wise for each tuple element.
#E.g. if the sequence filtered is (4,1), (2,6), (6,6),(3,5), then the sum of
#these tuples that is displayed is (4+2+6+3, 1+6+6+5) = (15, 18).
```