HW 5

Kevin Leary

PROBLEM 1

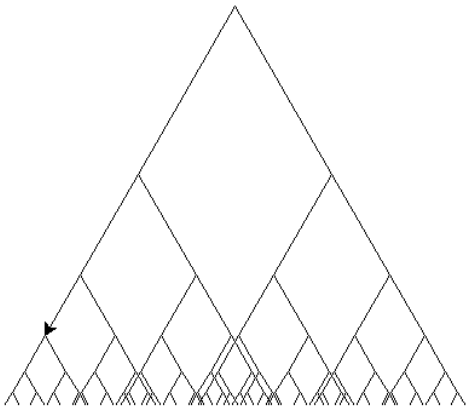
```
import turtle

#t_left = turtle.Turtle()
#t_right = turtle.Turtle()
t = turtle.Turtle()

#1A
def binary_tree(depth, length):
    """This function is supposed to return a Binary Tree with depth six
        It its drawing opposite on other levels
    """
    a = 60

    if depth <= 0:
                # base case
        return

    t.left(a)
    t.forward(length)
    t.right(a)
    binary_tree(depth-1, 0.6*length)
    t.left(a)
    t.penup()
    t.backward(length)
    t.pendown()

    t.right(a)
    t.forward(length)
    t.left(a)
    t.right(a)
    binary_tree(depth-1, 0.6*length)
    t.penup()
    t.backward(length)
    t.pendown()
    #t.right(a)
    #t.forward(length)
    return

#1B
```

```
def power_linear(x,n):
    """Uses a divide and conquer approach to compute powers
        where x is any number and n is a non negative integer
        so in the form X^n
    """
    try:
        if n == 0:          #base case
            return 1

        if n == 1:          #case if power = n which is X for all n
            return x

        if n%2 == 0:
            return power_linear(x, n/2) * power_linear(x, n/2)

        else:       #if its odd it will substract 1 then divide and continue
            return x * power_linear(x, (n-1)/2) * power_linear(x, (n-1)/2)
    except ValueError:
        print("Entered a wrong value for X or N")


def test_power():
    """Tests the cases of power_linear using testif()

    """
    testname = "test power_linear()"
    b = 0

    if power_linear(1, 0) == 1:
        if power_linear(7, 1) == 7:
            if power_linear(2, 7) == 128:
                b = 1

    return testif(b, testname)

#1C
def slice_sum(lst, begin, end):
    """Adds the elements of lst from 0 to end-1

    """
    try:
        if begin == end:
            return 0
        else:
            return lst[begin] + slice_sum(lst, begin+1, end)
```

```
    except IndexError:
        print("Error due to begin and end incompatibility")


def test_slice_sum():
    """Tests the slice_sum function for correctness

    """
    test_name = "test_slice_sum"
    lst = [1,2,3,4,5,6,7,8,9,10]
    b = 0
    if slice_sum(lst, 0, 4) == int(sum(lst[0:4])):
        if slice_sum(lst, 1, 6) == int(sum(lst[1:6])):
            if slice_sum(lst, 5, 10) == int(sum(lst[5:10])):
                if slice_sum(lst, 7, 8) == int(sum(lst[7:8])):
                    b = 1
    return testif(b, test_name)


def slice_sum_m(lst, begin, end):
    """Adds the elements of lst from 0 to end-1 using memoization

    """
    sum_dict = {}
    try:
        if begin == end:
            return 0
        else:
            value = lst[begin] + slice_sum(lst, begin+1, end)
        sum_dict = value
        return sum_dict
    except IndexError:
        print("Error due to begin and end incompatibility")


def test_slice_sum_m():
    """Tests the slice_sum_m function for correctness.

    """
    test_name = "test_slice_sum_m"
    lst = [1,2,3,4,5,6,7,8,9,10]
    b = 0
    if slice_sum_m(lst, 0, 4) == int(sum(lst[0:4])):
        if slice_sum_m(lst, 1, 6) == int(sum(lst[1:6])):
            if slice_sum_m(lst, 5, 10) == int(sum(lst[5:10])):
```

```
                if slice_sum_m(lst, 7, 8) == int(sum(lst[7:8])):
                    b = 1
    return testif(b, test_name)


#testif ----------
def testif(b, testname, msgOK="", msgFailed=""):
    """Function used for testing power_linear(x,n)
        param b: boolean, normallya  tested condition
        param testname: the test name
        param msgOK: string to be printed if b ==True
        param msgFailed: string to be printed if param b ==False
        returns b
    """
    if b:
        print("Sucess: " + testname + "; " + msgOK)
    else:
        print("Failed: " + testname + "; " + msgFailed)
    return b



#a = 120
# turn to get started
#t.penup()

#t.left(-120)
#t_right.right(60)
#t.pendown()


#1A - test
t.right(120)
binary_tree(6,160)

#1B - test
#test_power()

#1C - test
#test_slice_sum_m()
#test_slice_sum()
```

1A Output:



Squished but does it correctly

1B Output:

```
Sucess: test power_linear();
```

1C Output:

```
Sucess: test_slice_sum_m;
Sucess: test_slice_sum;
```

PROBLEM 2

```
import sys
import math

class PrimeSeq:

    __primes = list()       #instance attribute???

    def __init__(self, count):
        """ Default initalizer

        """
        self.count = count
```

```
    def __iter__(self):
        """ needs to have __iter__() to be for() compatable

        """

        return self

    def __next__(self):
        """ Aslo needs __next__() to be for compatable

        """
        #if len(self.__primes) <= self.count:
        """ if self.n <= self.count:
            for p in range(2, sys.maxsize**10):
                for i in range(2, p):
                    if p % i == 0:
                        break
                    else:

                        self.n +=1
                        return self.__primes.append(p) """

        if self.count > 2:
            for n in range(2, self.count):
                if self.__isprime(n):
                    n += 1
                    return n
        else:
            raise StopIteration

    def __isprime(self, n):
        """ Checks if the number is prime
            returns boolean
        """
        self.n = n

        for i in range(2, self.n):
            if self.n % i == 0:
                return False
        return True

#2B
def prime_gen(n):
    """takes an integer n >= 0 and produces the sequence of the first n prime
numbers.
```

```
    This generator is defined as a function that uses the yield keyword to output
a value

    """
    start = 2
    for i in range(start, int(math.sqrt(n)) + 1):
        if n % 1 == 0:
            break
        else:
            yield i



#2A

# primeseq = PrimeSeq(100)
# for p in primeseq:
#     print(p)

#2B
for p in prime_gen(10):
    print(p)
```

PROBLEM 3

```
import random
import itertools

def 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.
    """


    # for i in range(0, 10):

    #     print(str(tup) + " ")
    #tup = ()

    while True:
        a = random.randint(1, n)
```

```
        b = random.randint(1, n)
        tup = (a, b)
        yield tup



def main():

    for i in itertools.islice(gen_rndtup(7), 10):
        print(i)



main()
```

PROBLEM 3: OUPUT

```
(7, 5)
(1, 3)
(2, 1)
(2, 7)
(1, 6)
(3, 4)
(1, 4)
(7, 4)
(1, 2)
(5, 3)
```

PROBLEM 4

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys

def image_load(filename):
    return plt.imread(filename)


def image_gen(file1, file2, steps=30):
    """Generator for image arrays."""
    img1 = image_load(file1)     # load the two image files into ndarrays
    img2 = image_load(file2)
```

```
    if img1.shape != img2.shape:
        print("Error: the two images have different shapes.", file=sys.stderr)
        exit(2)

    # go from img1 to img2 than back to img1. s varies from 0 to 1 and then back
to 0:
    svalues = np.hstack([np.linspace(0.0, 1.0, steps), np.linspace(1.0, 0,
steps)])

    # construct now the list of images, so that we don't have to repeat that
later:
    images = [np.uint8(img1 * (1.0 - s) + img2 * s) for s in svalues]

    # get a  new image as a combination of img1 and img2
    while True:             # repeat all images in a loop
        for img in images:
           yield img

fig = plt.figure()
# create image plot and indicate this is animated. Start with an image.
im = plt.imshow(image_load("florida-keys-800-480.jpg"), interpolation='none',
animated=True)

# the two images must have the same shape:
imggen = image_gen("florida-keys-800-480.jpg", "Grand_Teton-800-480.jpg",
steps=30)

# updatefig is called for each frame, each update interval:
def updatefig(*args):
    global imggen
    img_array = next(imggen)     # get next image animation frame
    im.set_array(img_array)       # set it. FuncAnimation will display it
    return (im,)

# create animation object that will call function updatefig every 60 ms
ani = animation.FuncAnimation(fig, updatefig, interval=60, blit=False)
plt.title("Image transformation")
plt.show()
```