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

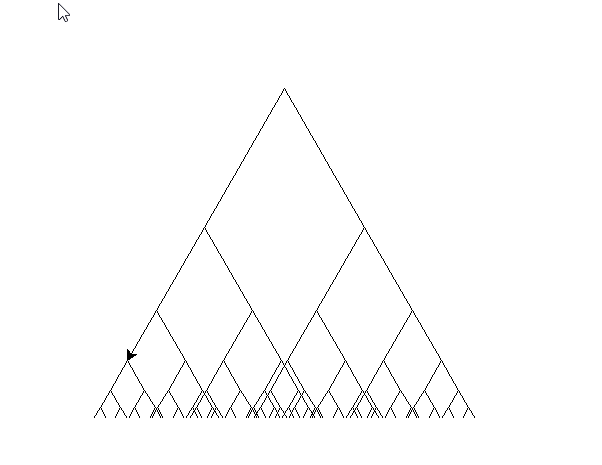
#test\_power()

#1C - test

#test\_slice\_sum\_m()

#test\_slice\_sum()

**1A Output:**

****

Squished but does it correctly

**1B Output:**

****

**1C Output:**

****

**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

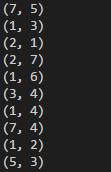
def main():

for i in itertools.islice(gen\_rndtup(7), 10):

print(i)

main()

**PROBLEM 3: OUPUT**

****

**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()