COMP7001 Introduction to Programming Portfolio Template Weeks 4-10

You should use this file as a template for the Turnitin submission of the Portfolio Exercises set in Week 4 onwards.

The solutions that you paste in here can be different to the ones originally uploaded to Replit.

# Academic Conduct

The contents of your portfolio must be your own, individual work. You should ensure that you are aware of the university rules on plagiarism and collusion.

# Portfolio Exercises

Put your submissions to the portfolio exercises here. Use a fixed width font such as Courier New to preserve the indentation of your code when you paste it in.

N.B. You must upload your submission as a Word document with the extension .doc, or .docx, or as an Open Office document (extension .odt, .doc, or .docx). Do **NOT**  upload documents with the extension .py, because they cannot be processed by Turnitin. Do **NOT** upload PDF documents, because these pose problems for us if we try to run your code.

# Week 4

## WK4EX8 [P] distance

import math

def distance(x1,y1,x2,y2):

dist = math.sqrt((x1 - x2)\*\*2 + (y1 - y2)\*\*2)

return dist

if \_\_name\_\_=="\_\_main\_\_":

x1 = (float(input('Enter value: ')))

y1 = (float(input('Enter value: ')))

x2 = (float(input('Enter value: ')))

y2 = (float(input('Enter value: ')))

equation = distance(x1,y1,x2,y2)

print(equation)

## WK4EX9 [P] count\_odds

def count\_odds(numlist):

odds = 0

for num in numlist:

if num%2 != 0:

odds +=1

return odds

if \_\_name\_\_=="\_\_main\_\_":

user\_input = input("Enter a list of values separated by comma: ")

split\_list = [int(n) for n in user\_input.split(",")]

num\_of\_odds = count\_odds(split\_list)

print(num\_of\_odds)

# Week 6

## WK6EX8 [P] words\_starting

# Function to return a list of words from a file

def words\_starting(file,lower,upper):

myfile = open(file, 'r')

list = []

# Logic to isolate index 0 from every word in file

for line in myfile:

words = line.split()

for word in words:

first\_letter = word[0]

# A word is saved in 'list', if its index 0, is in range between 'lower' and 'upper' parameters

if lower <= first\_letter <= upper:

list.append(word)

myfile.close()

return list

if \_\_name\_\_ == "\_\_main\_\_":

# '[0]' to recognise index 0 only, from user input

lower = str(input('Enter the lower letter: '))[0]

upper = str(input('Enter the upper letter: '))[0]

# Function call

letters = words\_starting('richmond.txt',lower,upper)

print('Words in richmond.txt starting with letters: ',letters)

## WK6EX9 [P] plurality\_winner

# Function that returns preferred candidates, called in 'plurality\_winner' function.

def count\_votes(filename,candidate):

score = 0

votes\_file = open(filename, 'r')

# Logic to isolate element 1 (most preferred) candidate in 'votes.txt'

for vote in votes\_file:

preference = vote.split()[1]

# Checks if most preferred candidate = specified candidate

if preference == candidate:

score += 1

votes\_file.close()

return score

# Function that returns the elections winners and their maximum scores only

def plurality\_winner(filename,candidates):

winner\_list = []

# Loop to count votes

for candi in candidates:

winner\_list.append(count\_votes(filename,candi))

# Identify maximum score among all candidates

max\_score = max(winner\_list)

# Returns winner identity and its max score

return [candidates[winner\_list.index(max\_score)],max\_score]

if \_\_name\_\_ == "\_\_main\_\_":

win = plurality\_winner('votes.txt',['Memphis','Nashville','Chattanooga','Knoxville'])

print('Plurality winner from votes.txt is ',win)

winner = plurality\_winner('votes2.txt',['Arthur','Betty','Clare'])

print('The winner from votes2.txt is ',winner)

# Week 7

## WK7EX10&11 [P] Marathon Progam

# Returns a list of dictionaries, where the keys define data about individual competitors

def read\_file(filename):

myfile = open('marathon.txt')

results = []

# Splits file’s elements separated by comma (of every row)

for line in myfile:

element = line.split(',')

id = element[0]

time = element[1]

firstname = element[2]

lastname = element[3]

# Stores participants' info in the corresponding dictionary key/section

results.append({'id': id, 'time': time, 'firstname': firstname, 'lastname': lastname})

myfile.close()

return results

# Returns stats for competitors in a specified time range

def get\_interval\_data(start\_secs, end\_secs, results):

count = 0

seconds\_sum = 0

# Turns participants 'time' to seconds

# 'time' extracted from dictionary created in read\_file() function

for competitor in results:

competitor\_time = get\_secs(competitor['time'])

# Counts participants who finished within the user-specified time range

if start\_secs <= competitor\_time <= end\_secs:

count += 1

# Accumlates time to calculate average

seconds\_sum += competitor\_time

if count != 0:

mean = seconds\_sum / count

else:

0

return {'count':count, 'mean':mean}

# Week 8

## EX8EX5&6 [P] ITriangle

# A program using measurements and coordinates of an isosceles triangle

import math

class Shape:

def \_\_init\_\_(self,xcen,ycen):

self.xcen = xcen

self.ycen = ycen

def move(self,dx,dy):

self.xcen += dx

self.ycen += dy

# ITriangle is a subclass of class 'Shape'

class ITriangle(Shape):

def \_\_init\_\_(self,xcen,ycen,base,height):

super().\_\_init\_\_(xcen,ycen)

self.base = base

self.height = height

# Area of triangle

def get\_area(self):

return self.base \* self.height / 2

# Perimeter of triangle

def get\_perimeter(self):

return self.base + math.sqrt(self.base\*\*2 + 4\*self.height\*\*2)

# Detects whether the coordinates entered are inside our triangle

def is\_inside(self,x,y):

distance\_x = x - self.xcen

distance\_y = y - (self.ycen - self.height/2)

# Returns True when, otherwise 'False':

condition\_1 = distance\_y + self.height > 0

condition\_2 = 2 \* self.height \* distance\_x - self.base \* distance\_y > 0

condition\_3 = 2 \* self.height \* distance\_x + self.base \* distance\_y < 0

return condition\_1 and condition\_2 and condition\_3

# Week 9

## WK9EX3 [P] Magic Squares

import numpy as np

# Function returns True if arr represents a magic square, otherwise False

def is\_magic\_square(arr):

# Counts rows and columns of arr

row, col = arr.shape

# If n rows != n columns, arr is NOT a matrix square

if row != col:

return False

# If sum of each column != sum of each row

if not np.all(arr.sum(axis=0) == arr.sum(axis=1)):

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

magic = np.array([[2,7,6],[9,5,1],[4,3,8]])

print("Applying is\_magic\_square to")

print(repr(magic))

print("Result = ",is\_magic\_square(magic))

non\_magic = np.array([[2,6,7],[9,5,1],[4,3,8]])

print("Applying is\_magic\_square to")

print(repr(non\_magic))

print("Result = ",is\_magic\_square(non\_magic))

# Week 10

## WK10EX2 [P] GE Turnout