Python Programming

Homework 4

Jordan Diaz

Problem 1

# This program strengthens skills in files and exceptions  
def ed\_append(filename, string):  
 *""" appends string to the end of the file, if file doesn't exist new file will be created with the filename. The  
 function returns the number of characters written to the file """* try:  
 file = open(filename, "a")  
 file.write(string)  
  
 return len(string)  
 except PermissionError:  
 print("There was a permission Error")  
 finally:  
 file.close()  
  
  
def ed\_read(filename, fr=0, to=-1):  
 *""" returns as a string the content of the file named filename, with file positions in the half-open range [from,  
 to). If to == -1 the content between from and the end of the file will be returned. If parameter to exceeds the  
 file length, then the function raises exception IndexError with a corresponding error message """* try:  
 file = open(filename, "r")  
 dumped\_text = file.read()  
 final\_str = ""  
 if to == -1:  
 end\_of\_range = len(dumped\_text)  
 elif to > fr:  
 end\_of\_range = to  
  
 for i in range(fr, end\_of\_range):  
 final\_str += dumped\_text[i]  
  
 return final\_str  
  
 except FileNotFoundError:  
 print("ERROR, File could not be found")  
 except PermissionError:  
 print("ERROR, Do not have permission to open file. May be used by another application")  
 except IndexError:  
 print("ERROR, parameter exceeds the file length")  
 finally:  
 file.close()  
  
  
def ed\_find(filename, search\_str):  
 *""" finds search\_str in the file named by filename and returns a list with index positions in the file text where  
 the string search\_str is located. E.g. it returns [4, 100] if the string was found at positions 4 and 100. it  
 returns [] if the string was not found. """* try:  
 file = open(filename, "r")  
 dumped\_text = file.read()  
  
 str\_to\_search = dumped\_text  
  
 final\_list = [i for i in range(len(str\_to\_search)) if str\_to\_search.startswith(search\_str, i)]  
  
 return final\_list  
  
 except FileNotFoundError:  
 print("ERROR, File could not be found")  
 except PermissionError:  
 print("ERROR, Do not have permission to open file. May be used by another application")  
 except IOError:  
 print("ERROR, There was an issue with I/O devices")  
 finally:  
 file.close()  
  
  
def ed\_replace(filename, search\_str, replace\_with, occurrence=-1):  
 *""" Replaces search\_str in the file named by filename with string replace\_with. If occurrence == -1,  
 then it replaces ALL occurrences. If occurrence>=0, then it replaces only the occurrence with index occurrence,  
 where 0 means the first, 1 means the second, etc. If the occurrence argument exceeds the actual occurrence index  
 in the file of that string, the function does not do the replacement. The function returns the number of times  
 the string was replaced. """* try:  
 file = open(filename, "r")  
 dumped\_text = file.read()  
  
 lst\_of\_occurrences = ed\_find(filename, search\_str)  
 if len(lst\_of\_occurrences) > 0:  
  
 str\_to\_get = dumped\_text[lst\_of\_occurrences[0]: lst\_of\_occurrences[0] + len(search\_str)]  
  
 times\_replaced = 0  
  
 if occurrence >= 0:  
 final\_text = dumped\_text[:lst\_of\_occurrences[occurrence]] + replace\_with + dumped\_text[len(search\_str) +  
 lst\_of\_occurrences[  
 occurrence]:]  
 times\_replaced = 1  
 elif occurrence == -1:  
 final\_text = dumped\_text.replace(str\_to\_get, replace\_with)  
 times\_replaced = len(lst\_of\_occurrences)  
  
 file2 = open(filename, "w")  
 file2.write(final\_text)  
  
 return times\_replaced  
 else:  
 print("Substring could not be found")  
  
 except FileNotFoundError:  
 print("ERROR, File could not be found")  
 except PermissionError:  
 print("ERROR, Do not have permission to open file. May be used by another application")  
 except IOError:  
 print("ERROR, There was an issue with I/O devices")  
 finally:  
 file.close()  
 file2.close()  
  
  
def testif(b, testname, msgOK="", msgFailed=""):  
 *""" Used for Unit Testing"""* if b:  
 print("Success: " + testname + "; " + msgOK)  
 else:  
 print("Failed: " + testname + "; " + msgFailed)  
 return b  
  
  
def test\_ed\_replace():  
 *""" Unit test for ed\_replace"""* fn = "test\_ed\_replace.txt"  
 ed\_append(fn, "bun lettuce cheese tomato patty bun")  
 testif(ed\_replace(fn, "lettuce", "", 0) == 1, "ed\_replace with occurrence")  
 testif(ed\_replace(fn, "bun", "") == 2, "ed\_replace without occurrence")  
 # File Contents should be cheese tomato patty  
  
  
def test\_ed\_find():  
 *""" Unit test for ed\_replace"""* fn = "test\_ed\_find.txt"  
 ed\_append(fn, "epicbyepicbyepicbyepicepic")  
 testif(ed\_find(fn, "epic") == [0, 6, 12, 18, 22], "ed\_find when found")  
 testif(ed\_find(fn, "calculator") == [], "ed\_find when not found")  
  
  
def main():  
 *""" Main Function works as intended when files do not exist yet"""* fn = "file.txt"  
  
 ed\_append(fn, "0123456789") # this will create a new file  
 ed\_append(fn, "0123456789") # the file content is: 01234567890123456789  
  
 print(ed\_read(fn, 3, 9)) # prints 345678. Notice that the interval excludes index to (9)  
 print(ed\_read(fn, 3)) # prints from 3 to the end of the file: 34567890123456789  
  
 lst = ed\_find(fn, "345")  
 print(lst) # prints [3, 13]  
 print(ed\_find(fn, "356")) # prints []  
  
 ed\_replace(fn, "345", "ABCDE", 1) # changes the file to 0123456789012ABCDE6789  
  
 ed\_replace(fn, "01", "popcorn") # changes the file to popcorn23456789popcorn2ABCDE6789  
  
 # Unit Testing  
 test\_ed\_replace()  
 test\_ed\_find()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal Session for problem 1

Text

Description automatically generated

Problem 2

# This program practices recursion  
from turtle import \*  
  
  
def draw\_leaf\_straight(level, length):  
 *""" draws a leaf-like that looks like figure 1-a using the turtle module."""* if level <= 0:  
 return  
  
 forward(length)  
  
 backward(length \* 0.6) # go to the lower half of the line  
  
 # Create a line to the left and reposition  
 left(45)  
 forward(length \* 0.3)  
 backward(length \* 0.3)  
 right(45)  
  
 # Go to the tip of the previous line and draw the left of the leaf  
 left(45)  
 draw\_leaf\_straight(level - 1, length \* 0.6)  
  
 # Center the cursor  
 right(90)  
  
 # Draw the right of the leaf  
 draw\_leaf\_straight(level - 1, length \* 0.6)  
 left(45)  
  
 # position to the right and make a line and come back, then center  
 right(45)  
 forward(length \* 0.3)  
 backward(length \* 0.3)  
 left(45)  
  
 forward(length \* 0.6) # go to the top of the first line  
  
 draw\_leaf\_straight(level - 1, length \* 0.6) # draw a leaf at the tip of the top  
  
 backward(length)  
  
  
def strB(n, base=10):  
 *""" Converts a non-negative int value n to a string representation of n in the given base. The base parameter is  
 an int between 2 and 26. For digits greater than 9 use letters 'A'-'Z' """* characters = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 if n < base:  
 return characters[n]  
 else:  
 return str(strB(n // base, base)) + characters[n % base]  
  
  
def testif(b, testname, msgOK="", msgFailed=""):  
 *""" Used for unit testing"""* if b:  
 print("Success: " + testname + "; " + msgOK)  
 else:  
 print("Failed: " + testname + "; " + msgFailed)  
 return b  
  
  
def Cnk\_m(n, k):  
 *""" Computing the value of the binomial coefficient using the memoization technique taught in class"""* binomial\_coefficent\_cache = {}  
 if (n, k) in binomial\_coefficent\_cache:  
 return Cnk\_m((n, k)[0], (n, k)[1])  
  
 if k > n:  
 result = 0  
 elif k == n or k == 0:  
 result = 1  
 else:  
 result = Cnk\_m(n - 1, k - 1) + Cnk\_m(n - 1, k)  
  
 # Cache  
 binomial\_coefficent\_cache[(n, k)] = result  
 return result  
  
  
def make\_pairs(seq1, seq2):  
 *"""takes as parameters two lists, seq1 and seq2, and that returns a list with all tuples (x, y) where x is in  
 seq1 and y is the matching element in seq2, at the same index as x.Function make\_pairs stops once it reaches the  
 end of the shorter sequence. """* if len(seq1) == 0 or len(seq2) == 0:  
 return []  
 return ([(seq1[0], seq2[0])]) + (make\_pairs(seq1[1:], seq2[1:]))  
  
  
def main():  
 # Part a  
 clearscreen()  
 left(90)  
 speed(0)  
 delay(0)  
 draw\_leaf\_straight(6, 120)  
 done()  
  
 # Part b  
 print()  
 print("Test for strB")  
  
 testif(strB(123, base=16) == "7B", "Test 1")  
 testif(strB(1234, base=16) == "4D2", "Test 2")  
 testif(strB(123456789, base=26) == "AA44A1", "Test 3")  
 testif(strB(100, base=2) == "1100100", "Test 4")  
  
 print()  
 print("Tests for Cnk\_m")  
  
 testif(Cnk\_m(10, 10) == 1, "Test 5")  
 testif(Cnk\_m(50, 5) == 2118760, "Test 6")  
 testif(Cnk\_m(10, 3) == 120, "Test 7")  
 testif(Cnk\_m(9, 12) == 0, "Test 8")  
  
 print()  
 print("Tests for make\_pairs")  
  
 testif(make\_pairs([1, 2, 3], [4, 5, 6]) == [(1, 4), (2, 5), (3, 6)], "Test 9")  
 testif(make\_pairs([1, 2, 3], [4, 5]) == [(1, 4), (2, 5)], "Test 10")  
 testif(make\_pairs([1, 2, 3], [4, 5, 6, 7, 8, 9]) == [(1, 4), (2, 5), (3, 6)], "Test 10")  
 testif(make\_pairs([], [4, 5, 6, 7, 8, 9]) == [], "Test 11")  
 testif(make\_pairs([1, 2, 3], []) == [], "Test 12")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal Session for problem 2

A picture containing text, screenshot

Description automatically generated

Text

Description automatically generated