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

```
345678
34567890123456789
[3, 13]
[]
Success: ed_replace with occurrence;
Success: ed_replace without occurrence;
Success: ed_find when found;
Success: ed_find when not found;
```

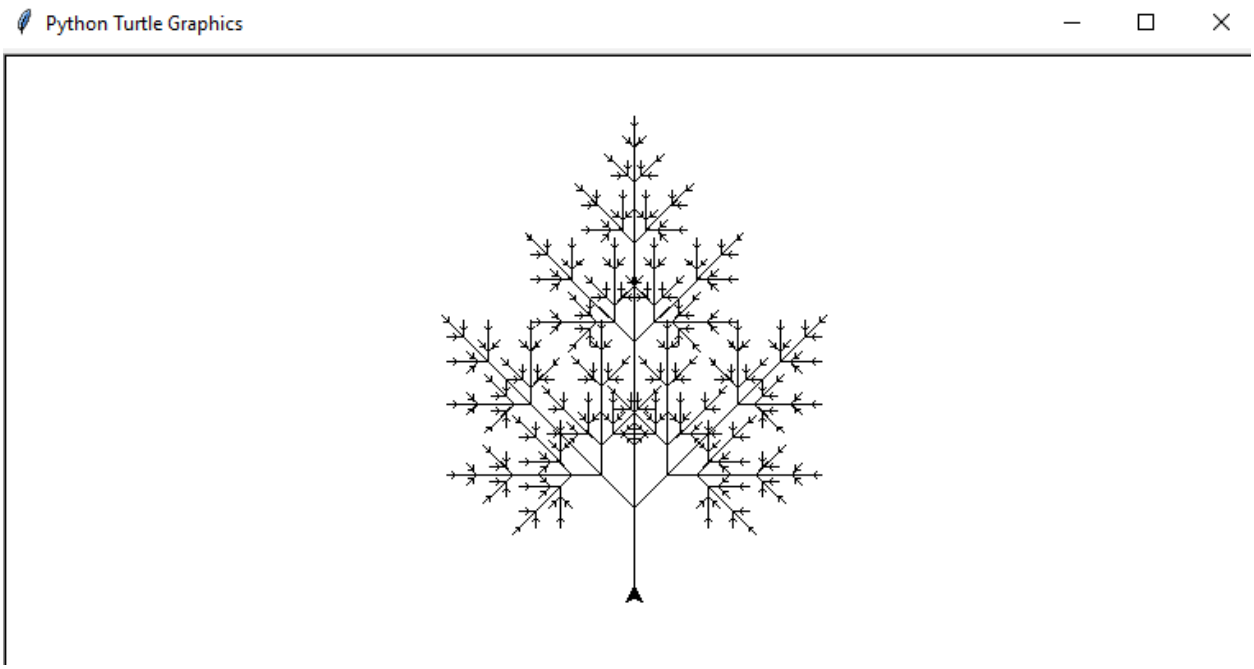
Problem 2

```
# 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



```
Test for strB
Success: Test 1;
Success: Test 2;
Success: Test 3;
Success: Test 4;

Tests for Cnk_m
Success: Test 5;
Success: Test 6;
Success: Test 7;
Success: Test 8;

Tests for make_pairs
Success: Test 9;
Success: Test 10;
Success: Test 10;
Success: Test 11;
Success: Test 12;
```