ME 5194: HW 6

Adrian Bakhtar

CHAPTER 14

Exercise 14.1

Write a function called *sed* that takes as arguments a pattern string, a replacement string, and two filenames; it should read the first file and write the contents into the second file (creating it if necessary). If the pattern string appears anywhere in the file, it should be replaced with the replacement string.

If an error occurs while opening, reading, writing or closing files, your program should catch the exception, print an error message, and exit.

In [2]:
```
import os


def file1_check(file1):
     try:
        if not os.path.exists(file1):
            print(f"{file1} does not exist. Double check the filename/path!")
            return False
        if not os.path.isfile(file1):
            print(f"{file1} is not a file. Double check the filename/path!")
            return False
        return True
     except:
        print(f"An error occurred while opening {file1}. Double check the filename/path!")
        return False

def file2_check(file2):
    if os.path.exists(file2):
        user_resp = input(f"{file2} already exists and would be overwritten.\n\n Would you like to enter a new filename? (y/n):\t

        if user_resp.upper() == 'Y' or user_resp == 'y':
            new_file2_name = input('Enter a new name for the second file without the filetype: ')
            new_file2 = new_file2_name + '.txt'

        elif user_resp.upper() == 'N' or user_resp == 'n':
            print(f'{file2} will be overwritten.')
            new_file2 = file2
    else: new_file2 = file2
    return new_file2


def sed(pattern, replace, file1, file2):

    f1_check = file1_check(file1)
    if f1_check == False: return
    f1 = open(file1, 'r')

    final_file2 = file2_check(file2)
    f2 = open(final_file2, 'w')

    for ln in f1:
        rep_ln = ln.replace(pattern, replace)
        f2.write(rep_ln)

    f1.close()
    f2.close()

pattern1 = 'little star'
replace1 = 'big red car'

file_test1 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products/twinkletwiklelittlestar.txt'
file_test2 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products/twinkle bigRedCar.txt'

sed(pattern1, replace1, file_test1, file_test2)
```

```
C:/Users/Adrian Bakhtar/Documents/Smart Products/twinkletwiklelittlestar.txt does not exist. Double check the filename/path!
```

In [3]:
```
file_test1 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products'
file_test2 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products/twinkle big red car.txt'

sed(pattern1, replace1, file_test1, file_test2)
```

```
C:/Users/Adrian Bakhtar/Documents/Smart Products is not a file. Double check the filename/path!
```

In [63]:
```
file_test1 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products/twinkle twikle little star.txt'
file_test2 = 'C:/Users/Adrian Bakhtar/Documents/Smart Products/bigRedCar.txt'

sed(pattern1, replace1, file_test1, file_test2)
```

CHAPTER 15

Exercise 15.1

Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number.

Instantiate a Circle object that represents a circle with its center at (150, 100) and radius 75.

Write a function named *point_in_circle* that takes a Circle and a Point and returns True if the Point lies in or on the boundary of the circle.

Write a function named *rect_in_circle* that takes a Circle and a Rectangle and returns True if the Rectangle lies entirely in or on the boundary of the circle.

Write a function named *rect_circle_overlap* that takes a Circle and a Rectangle and returns True if any of the corners of the Rectangle fall inside the Circle. Or as a more challenging version, return True if any part of the Rectangle falls inside the Circle.

In [55]:
```
#Point and Rectangle Classes from Book
from __future__ import print_function, division


class Point:
    """Represents a point in 2-D space.
    attributes: x, y
    """
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

def print_point(p):
    """Print a Point object in human-readable format."""
    print('(%g, %g)' % (p.x, p.y))


class Rectangle:
    """Represents a rectangle.
    attributes: width, height, corner.
    """


def find_center(rect):
    """Returns a Point at the center of a Rectangle.
    rect: Rectangle
    returns: new Point
    """
    p = Point()
    p.x = rect.corner.x + rect.width/2.0
    p.y = rect.corner.y + rect.height/2.0
    return p


def grow_rectangle(rect, dwidth, dheight):
    """Modifies the Rectangle by adding to its width and height.
    rect: Rectangle object.
    dwidth: change in width (can be negative).
    dheight: change in height (can be negative).
    """
    rect.width += dwidth
    rect.height += dheight


blank = Point()
blank.x = 150
blank.y = 110


box = Rectangle()
box.width = 10.0
box.height = 20.0
box.corner = Point()
box.corner.x = 100.0
box.corner.y = 80.0
```

In [56]:
```
import math

#My function to get the points of the rectangle's corners
def rect_corners(rect_obj):
    p1 = rect_obj.corner
    p2 = Point(p1.x + rect_obj.width, p1.y)
    p3 = Point(p1.x, p1.y + rect_obj.height)
    p4 = Point(p1.x + rect_obj.width, p1.y + rect_obj.height)

    return [p1, p2, p3, p4]

class Circle:

    def __init__(self, center, radius):
        self.center = center
        self.radius = radius

    def point_in_circle(self, point_obj):
        total_dist = math.dist(self.center, (point_obj.x, point_obj.y))
        return total_dist <= self.radius

    def rect_in_circle(self, rect_obj):
        for corner in rect_corners(rect_obj):
            if not self.point_in_circle(corner):
                return False
        return True





c = Circle((150,100), 75)

point_in_circle = c.point_in_circle(blank)
rect_in_circle = c.rect_in_circle(box)

print(point_in_circle)
print(rect_in_circle)
```

```
True
True
```

In [58]:
```

blank = Point()
blank.x = 10
blank.y = 20


box = Rectangle()
box.width = 11.0
box.height = 2.0
box.corner = Point()
box.corner.x = 1000.0
box.corner.y = 1000.0


c = Circle((150,100), 75)

point_in_circle = (c.point_in_circle(blank))

rect_in_circle = c.rect_in_circle(box)

print(point_in_circle)
print(rect_in_circle)
```

```
False
False
```

CHAPTER 16

Exercise 16.2

The datetime module provides time objects that are similar to the Time objects in this chapter, but they provide a rich set of methods and operators. Read the documentation at http: // docs. python. org/ 3/ library/ datetime. html

1. Use the datetime module to write a program that gets the current date and prints the day of the week.
2. Write a program that takes a birthday as input and prints the user’s age and the number of days, hours, minutes and seconds until their next birthday.
3. For two people born on different days, there is a day when one is twice as old as the other. That’s their Double Day. Write a program that takes two birth dates and computes their Double Day.

In [59]:
```
from datetime import datetime


current_date = datetime.today()
day_of_week = current_date.strftime('%A')
print(f"Today's Date: {current_date}\nDay of Week: {day_of_week}")
```

```
Today's Date: 2023-02-20 00:10:36.663354
Day of Week: Monday
```

In [60]:
```
from datetime import datetime, timedelta

def get_bday():
        try:
            birthday = input("Please enter your birthday in mm/dd/yyyy format: ")
            birthday_date = datetime.strptime(birthday, '%m/%d/%Y').date()
            return birthday_date
        except ValueError:
            print("Please make sure to enter your birthday in mm/dd/yyyy format!")

def next_bday_countdown(bday):
    current_date = datetime.today()
    next_bday = datetime(current_date.year, bday.month, bday.day)

    if next_bday < current_date:
        next_bday = datetime(current_date.year+1, bday.month, bday.day)

    time_to_birthday = next_bday - current_date
    total_seconds = int(time_to_birthday.total_seconds())

    days, remaining_seconds = divmod(total_seconds, 60*60*24)
    hours, remaining_seconds = divmod(remaining_seconds, 60*60)
    minutes, seconds = divmod(remaining_seconds, 60)

    print(f"Time until next birthday: {days} days, {hours} hours, {minutes} minutes, {seconds} seconds")


birthday = get_bday()
age = datetime.now().date().year - birthday.year
print(f"Age:\t{age} years old.\n")
next_bday_countdown(birthday)
```

```
Please enter your birthday in mm/dd/yyyy format: 10/27/2000
Age:    23 years old.

Time until next birthday: 248 days, 23 hours, 49 minutes, 17 seconds
```

In [61]:
```
from datetime import datetime

def get_double_day():
    # Prompt the user to enter the two birth dates
    bday1_str = input("Enter the first person's birthday (mm/dd/yyyy):\t")
    bday1 = datetime.strptime(bday1_str, "%m/%d/%Y")

    bday2_str = input("Enter the second person's birthday (mm/dd/yyyy):\t")
    bday2 = datetime.strptime(bday2_str, "%m/%d/%Y")

    # Calculate the double day
    day1 = min(bday1, bday2)
    day2 = max(bday1, bday2)
    double_day = day2 + (day2 - day1)
    print(f"The Double Day is {double_day.strftime('%m/%d/%Y')}")


get_double_day()
```

```
Enter the first person's birthday (mm/dd/yyyy): 10/27/2000
Enter the second person's birthday (mm/dd/yyyy):        05/14/2001
The Double Day is 11/29/2001
```

CHAPTER 17

Exercise 17.2

This exercise is a cautionary tale about one of the most common, and difficult to find, errors in Python. Write a definition for a class named Kangaroo with the following methods:

1. An **init** method that initializes an attribute named pouch_contents to an empty list.
2. A method named put_in_pouch that takes an object of any type and adds it to pouch_contents.
3. A **str** method that returns a string representation of the Kangaroo object and the contents of the pouch. Test your code by creating two Kangaroo objects, assigning them to variables named kanga and roo, and then adding roo to the contents of kanga's pouch.

In [35]:
```
#MY FUNCTION
class Kangaroo:
    def __init__(self):
        self.pouch_contents = []

    def put_in_pouch(self, obj):
        self.pouch_contents.append(obj)

    def __str__(self):
        return f"Kangaroo Pouch contents: {self.pouch_contents}"


kanga = Kangaroo()
roo = Kangaroo()

kanga.put_in_pouch("wallet")
kanga.put_in_pouch(roo)

print(f'kanga: {kanga}')
print(f'roo: {roo}')
```

```
kanga: Kangaroo Pouch contents: ['wallet', <__main__.Kangaroo object at 0x000001E3A9E359D0>]
roo: Kangaroo Pouch contents: []
```

In [36]:
```
#BAD KANGAROO

class Kangaroo:
    """A Kangaroo is a marsupial."""

    def __init__(self, name, contents=[]):
        """Initialize the pouch contents.
        name: string
        contents: initial pouch contents.
        """
        self.name = name
        self.pouch_contents = contents

    def __str__(self):
        """Return a string representaion of this Kangaroo.
        """
        t = [ self.name + ' has pouch contents:' ]
        for obj in self.pouch_contents:
            s = '    ' + object.__str__(obj)
            t.append(s)
        return '\n'.join(t)

    def put_in_pouch(self, item):
        """Adds a new item to the pouch contents.
        item: object to be added
        """
        self.pouch_contents.append(item)


kanga = Kangaroo('Kanga')
roo = Kangaroo('Roo')
jack = Kangaroo('Jack')

kanga.put_in_pouch('wallet')
kanga.put_in_pouch('car keys')
kanga.put_in_pouch(roo)

print(kanga)
print(roo)
print(jack)
```

```
Kanga has pouch contents:
    'wallet'
    'car keys'
    <__main__.Kangaroo object at 0x000001E3A9F06190>
Roo has pouch contents:
    'wallet'
    'car keys'
    <__main__.Kangaroo object at 0x000001E3A9F06190>
Jack has pouch contents:
    'wallet'
    'car keys'
    <__main__.Kangaroo object at 0x000001E3A9F06190>
```

In [37]:
```
#GOOD KANGAROO

class Kangaroo:
    """A Kangaroo is a marsupial."""

    def __init__(self, name, contents=[]):
        """Initialize the pouch contents.
        name: string
        contents: initial pouch contents.
        """
        # The problem is the default value for contents.
        # Default values get evaluated ONCE, when the function
        # is defined; they don't get evaluated again when the
        # function is called.

        # In this case that means that when __init__ is defined,
        # [] gets evaluated and contents gets a reference to
        # an empty list.

        # After that, every Kangaroo that gets the default
        # value gets a reference to THE SAME list.  If any
        # Kangaroo modifies this shared list, they all see
        # the change.

        # The next version of __init__ shows an idiomatic way
        # to avoid this problem.
        self.name = name
        self.pouch_contents = contents

    def __init__(self, name, contents=None):
        """Initialize the pouch contents.
        name: string
        contents: initial pouch contents.
        """
        # In this version, the default value is None.  When
        # __init__ runs, it checks the value of contents and,
        # if necessary, creates a new empty list.  That way,
        # every Kangaroo that gets the default value gets a
        # reference to a different list.

        # As a general rule, you should avoid using a mutable
        # object as a default value, unless you really know
        # what you are doing.
        self.name = name
        if contents == None:
            contents = []
        self.pouch_contents = contents

    def __str__(self):
        """Return a string representaion of this Kangaroo.
        """
        t = [ self.name + ' has pouch contents:' ]
        for obj in self.pouch_contents:
            s = '    ' + object.__str__(obj)
            t.append(s)
        return '\n'.join(t)

    def put_in_pouch(self, item):
        """Adds a new item to the pouch contents.
        item: object to be added
        """
        self.pouch_contents.append(item)


kanga = Kangaroo('Kanga')
roo = Kangaroo('Roo')
kanga.put_in_pouch('wallet')
kanga.put_in_pouch('car keys')
kanga.put_in_pouch(roo)

print(kanga)
print(roo)
```

```
Kanga has pouch contents:
    'wallet'
    'car keys'
    <__main__.Kangaroo object at 0x000001E3A9F06100>
Roo has pouch contents:
```

After looking at both the bad and good kangaroo files and comparing them to my own, I noticed a few things. Based off of my initialization, I am not giving the user the option to name the Kangaroo object or pass any existing pouch_contents. I also had a hard time understanding why the error was happening. It would have made more sense to me if the other Kangaroo objects had the same items in the pouch if they were created after the items were added:

- kanga = Kangaroo('Kanga')

- kanga.put_in_pouch('wallet')
- kanga.put_in_pouch('car keys')
- kanga.put_in_pouch(roo)
- roo = Kangaroo('Roo')

but this was not the case. After searching for a more involved explanation, it finally clicked for me when I read that default argument values for the **init** method are only evaluated once, not every time that the method is called. When the contents = [] was evaluated, contents was asigned a reference to an empty list. So if other instances also had contents = [], they would be referencing the same empty list object. I was thinking that each

This along with the fact that lists are mutable objects, so if any instance of the class modifies this list, it is modified for all instances. This is why None was used as the correction, since it is an immutable object.

CHAPTER 18

Exercise 18.2

Write a Deck method called deal_hands that takes two parameters, the number of hands and the number of cards per hand. It should create the appropriate number of Hand objects, deal the appropriate number of cards per hand, and return a list of Hands.

In [38]:
```python
#Including the card code from the book for this problem
import random

class Card:
    suit_names = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
    rank_names = [None, 'Ace', '2', '3', '4', '5', '6', '7','8', '9', '10', 'Jack', 'Queen', 'King']

    def __init__(self, suit=0, rank=2):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        return '%s of %s' % (Card.rank_names[self.rank], Card.suit_names[self.suit])

    def __lt__(self, other):
        t1 = self.suit, self.rank
        t2 = other.suit, other.rank
        return t1 < t2

class Deck:
    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)
    def __str__(self):
        res = []
        for card in self.cards:
            res.append(str(card))
        return '\n'.join(res)

    def pop_card(self):
        return self.cards.pop()

    def add_card(self, card):
        self.cards.append(card)

    def shuffle(self):
        random.shuffle(self.cards)

    def move_cards(self, hand, num):
        for i in range(num):
            hand.add_card(self.pop_card())

    ##MY FUNCTION TO DEAL HANDS
    def deal_hands(self, num_hands, cards_per_hand):
        hands = []

        for i in range(num_hands):
            hand = Hand(f'Hand {i+1}')
            for j in range(cards_per_hand):
                hand.add_card(self.pop_card())
            hands.append(hand)
        return hands


class Hand(Deck):

    def __init__(self, label=''):
        self.cards = []
        self.label = label

    ##MY FUNCTION TO BETTER SEE THE CARDS IN EACH HAND
    def print_hand(self, cards_per_hand):
        count = 0
        for card in self.cards:
            print(card)
            count += 1
            if count % cards_per_hand == 0:
                print('\n')

deck = Deck()
deck.shuffle()
hands = deck.deal_hands(4, 5)

for cards in hands:
    print(f"{cards.label}:")
    cards.print_hand(5)
```

```
Hand 1:
7 of Diamonds
2 of Spades
2 of Hearts
7 of Clubs
Ace of Spades


Hand 2:
3 of Hearts
Ace of Clubs
6 of Clubs
King of Diamonds
7 of Hearts


Hand 3:
Jack of Diamonds
5 of Clubs
King of Clubs
10 of Spades
Queen of Clubs


Hand 4:
8 of Hearts
Queen of Hearts
6 of Diamonds
3 of Spades
6 of Spades
```