This exam is based on Chapter 7 Arrays, Chapter 10 Classes, and Chapter 13 TKINTER.

**There are 7 print screens, each worth 14.2%.**

**Project #1 (Arrays)**

Create a 1-D list with the numbers 1-10 and get the **sum** and **average** of the numbers.

**#1 Print screen the output with the code below here.**



Code:

list = [1,2,3,4,5,6,7,8,9,10]

total=0

for i in list:

total+=i

avg = total / 10

print(total)

print(avg)

In the 1-D list retrieve only the lucky number 7.

Val

**#2 Print screen the output with the code below here.**



Code:

list = [1,2,3,4,5,6,7,8,9,10]

val = list[6]

print(val)

Create a 2-D list with the numbers 1-10 and retrieve the lucky number 7. Below is an example of a 2-d list.

Table

Description automatically generated

**#3 Print screen the output with the code below here.**



Code:

list = [[1,2,3,4,5],[6,7,8,9,10]]

val = list[1][1]

print(val)

Create a 3-D list with the numbers 1-10 and retrieve the number 10.

**#4 Print screen the output with the code below here.**



Code:

list = [[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]],[[13,14,15]]]

val = list[1][1][0]

print(val)

**Project #2 (Classes).**

**Question**: Create a class called "Rectangle" that represents a rectangle shape. The class should have the following attributes and methods:

**Attributes**:

* width: representing the width of the rectangle.
* height: representing the height of the rectangle.

**Methods**:

* set\_dimensions(width, height): sets the width and height of the rectangle.
* get\_area(): returns the area of the rectangle.
* get\_perimeter(): returns the perimeter of the rectangle.

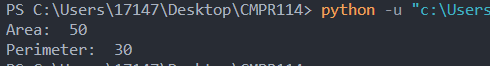
Write the code for the Rectangle class that implements the above requirements. Additionally, create an instance of the class, set its dimensions to 5 and 10, and print the area and perimeter of the rectangle.

Your code should output the following:

Area: 50

Perimeter: 30

**#5 Print screen the output with the code below here.**



Code:

class rectangle():

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def get\_area(self):

return self.width \* self.height

def get\_perimeter(self):

return (self.width \* 2) + (self.height \* 2)

obj = rectangle(5, 10)

print("Area: ", obj.get\_area())

print("Perimeter: ", obj.get\_perimeter())

**Project #3 (Classes).**

**Question**: Create a class called "BankAccount" that represents a bank account. The class should have the following attributes and methods:

**Attributes**:

* account\_number: representing the unique account number of the bank account.
* balance: representing the current balance of the bank account.
* owner: representing the owner of the bank account.

**Methods**:

* deposit(amount): adds the specified amount to the balance of the account.
* withdraw(amount): subtracts the specified amount from the balance of the account.

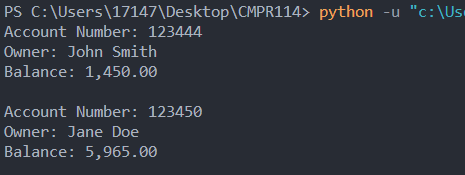
Make sure to check if the withdrawal amount is greater than the balance and handle that case appropriately.

* get\_balance(): returns the current balance of the account.
* display\_account\_info(): prints the account number, owner, and balance of the account.

Additionally, create two instances of the BankAccount class with different account numbers, owners, and initial balances. Perform some deposit and withdrawal operations on both accounts and display their updated information.

Write the code for the BankAccount class and the required operations on the accounts.

**#6 Print screen the output with the code below here.**



Code:

class bankaccount():

def \_\_init\_\_ (self, acctno, owner, balance):

self.acctno = acctno

self.owner = owner

self.balance = balance

def deposit(self, amount):

self.balance+=amount

def withdraw(self, amount):

if self.balance >= amount:

self.balance-=amount

else:

print("Insufficient funds.")

def get\_balance(self):

return self.balance

def display\_account\_info(self):

info = f"""Account Number: {self.acctno}\nOwner: {self.owner}\nBalance: {self.balance:,.2f}\n"""

return info

person1 = bankaccount(123444, "John Smith", 1000.00)

person1.deposit(200)

person1.deposit(400)

person1.withdraw(150)

print(person1.display\_account\_info())

person2 = bankaccount(123450, "Jane Doe", 5500.00)

person2.deposit(300)

person2.deposit(695)

person2.withdraw(530)

print(person2.display\_account\_info())

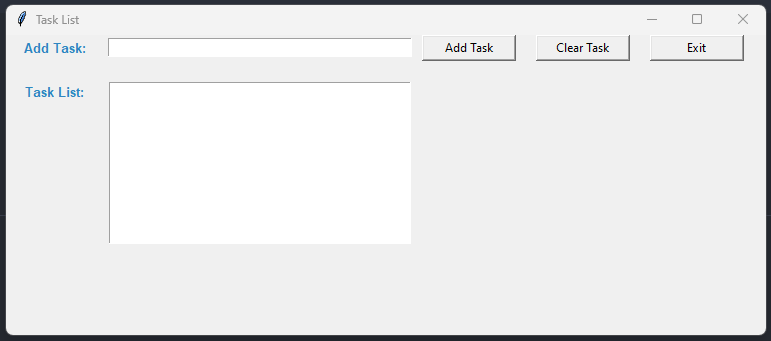
**Project #4 (TKINTER GUI).**

**Exercise**: Create a ***to-do list application*** using Tkinter. The application should have a window with an entry field to input tasks and an “Add Task" button to add the task to the list (use a **list** **view** control to display the lists). Below the entry field and button, display the list of tasks as labels. Each task label should have a checkbox next to it, allowing the user to mark a task as completed. Finally, add a "Delete Completed" button to remove all completed tasks from the list.

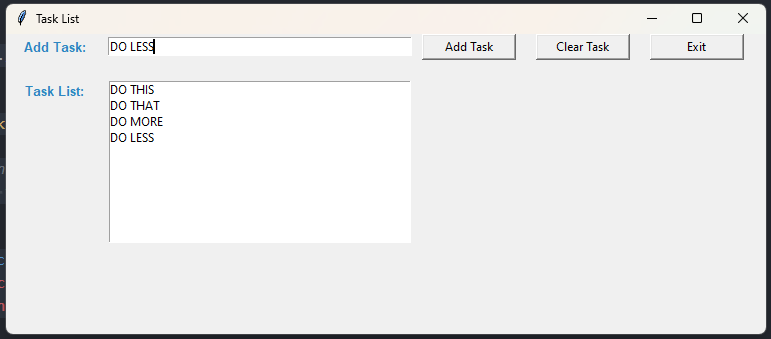
Write the code for the to-do list application using Tkinter.

**#7 Print screen the output (GUI) interface with the code below here. Please provide print screens for each step, indicating the program is complete. Multiple print screens are encouraged.**

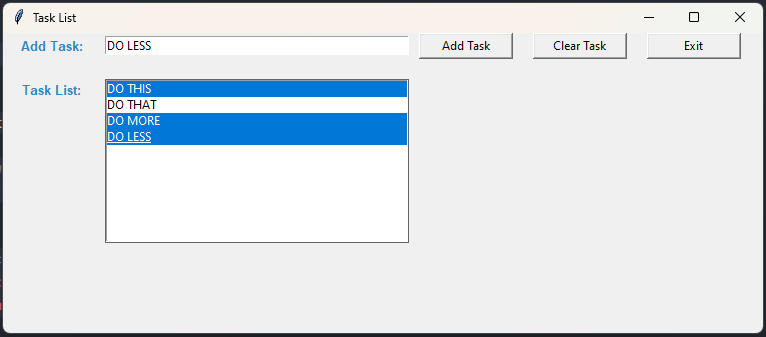
Initial Screen

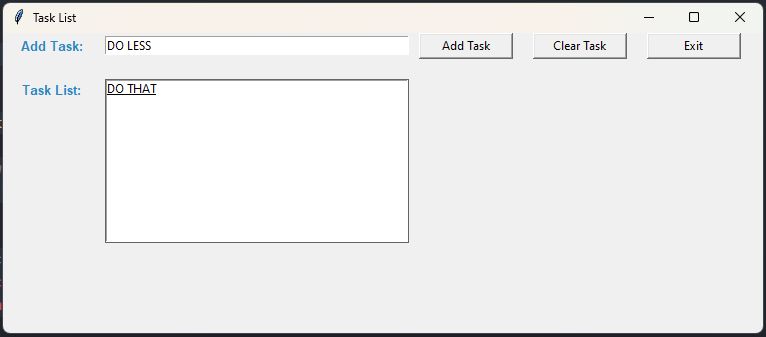
****

Added Tasks



Clear Task Selection





Code:

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

def add\_task():

tasks=[]

tasks.append(task.get())

for i in tasks:

print(i)

lbox.insert(tk.END, i)

#Clear out task entry textbox

txttask.delete(0, tk.END)

def clear\_task():

selection = lbox.curselection()

for index in selection[::-1]:

lbox.delete(index)

def exit():

win.quit()

win.destroy()

win = tk.Tk()

win.title('Task List')

win.geometry("760x300")

# Input

lbltask = tk.Label(win, text = "Add Task: ", font="Arial 10 bold", width=12, fg="#2E86C1")

lbltask.grid(column=0, row=0) #Label widge

task = tk.StringVar() #Manage the Entry widget

txttask = tk.Entry(win, width=50, textvariable=task)

txttask.grid(column=1,row=0, ipadx = 0, padx = 0, pady=3, sticky="w")

btnadd = tk.Button(win, text="Add Task", command=add\_task, width=12)

btnadd.grid(column=2, row=0, padx=10)

btnclear = tk.Button(win, text="Clear Task", command=clear\_task, width=12)

btnclear.grid(column=3, row=0, padx=10)

btnexit = tk.Button(win, text="Exit", command=quit, width=12)

btnexit.grid(column=4, row=0, padx=10)

txttask.focus\_set()

# Listbox

lbllist = tk.Label(win, text = "Task List: ", font="Arial 10 bold", width=12, fg="#2E86C1")

lbllist.grid(column=0, row=2, sticky="ne", pady=20) #Label widge

lbox = tk.Listbox(win, width=50, selectmode="multiple", yscrollcommand="Yes")

lbox.grid(column=1, row=2, pady=20)

win.mainloop()

**Submit this document to Module 5 Quiz.**