Objectives

* Inheritance and TKINTER

**There are 7 Exercises, each worth 14.2%.**

**Inheritance** allows a new class to extend an existing class. The new class **inherits** the members of the class it **extends**. Inheritance allows to inheritance of attributes and methods from a parent class to a child class or classes.

These examples demonstrate the concept of inheritance in Python, where classes inherit properties and methods from their parent class(es) and can also override or add new functionality.

**Project #1**

A screen shot of a computer program

Description automatically generated

**Project #2**

A screen shot of a computer program

Description automatically generated

**Project #3**

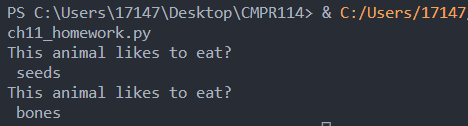
An object inherited from two different classes.

Text

Description automatically generated

**Challenge Exercise #1:** continuing from project #3, add the 3rd animal and print screen the results with the code below:

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



Code:

class AnimalType:

def eats(self):

print("This animal likes to eat? ")

class rabbits(AnimalType):

def munch(self):

print(" carrots")

class birds(AnimalType):

def munch1(self):

print(" seeds ")

class dogs(AnimalType):

def munch(self):

print(" bones")

BirdObject = AnimalType()

BirdObject = birds()

BirdObject.eats()

BirdObject.munch1()

DogObject = dogs()

DogObject.eats()

DogObject.munch()

**Project #4:** this example is using the super keyword to inherit attributes and arguments.

Graphical user interface, text, application

Description automatically generatedText, letter

Description automatically generated

\_\_init\_\_ method initializes object attributes so they can be shared throughout the program

Entering information and passing the attributes to parameters. **(Enter lines 13-16 and 30-33)**

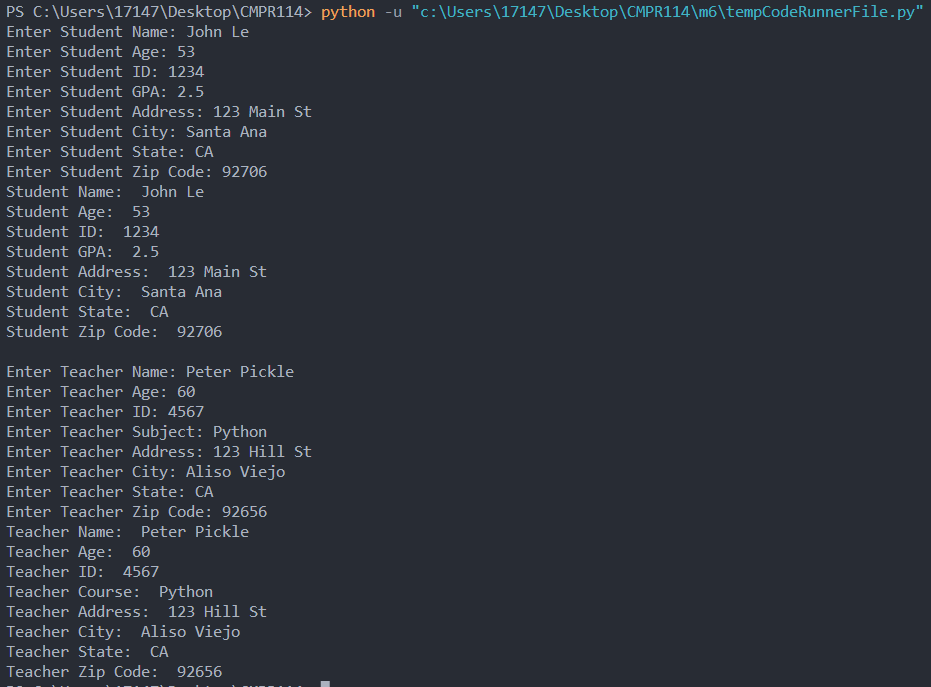
Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

**Challenge Exercise #2:** Continuing from project #4 add to the super constructor (see line 23) and add the address, city, state, and zip code) for the Student and Teacher. Print screen below.

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



Code:

class person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

class student(person):

def \_\_init\_\_(self, name, age, studentid, gpa, address, city, state, zipcode):

#get input

name = input("Enter Student Name: ")

age = input("Enter Student Age: ")

studentid = input("Enter Student ID: ")

gpa = input("Enter Student GPA: ")

address = input("Enter Student Address: ")

city = input("Enter Student City: ")

state = input("Enter Student State: ")

zipcode = input("Enter Student Zip Code: ")

super().\_\_init\_\_(name, age)

self.studentid = studentid

self.gpa = gpa

self.address = address

self.city = city

self.state = state

self.zipcode = zipcode

class teacher(person):

def \_\_init\_\_(self, name, age, teacherid, subject, address, city, state, zipcode):

name = input("Enter Teacher Name: ")

age = input("Enter Teacher Age: ")

teacherid = input("Enter Teacher ID: ")

subject = input("Enter Teacher Subject: ")

address = input("Enter Teacher Address: ")

city = input("Enter Teacher City: ")

state = input("Enter Teacher State: ")

zipcode = input("Enter Teacher Zip Code: ")

super().\_\_init\_\_(name, age)

self.teacherid = teacherid

self.subject = subject

self.address = address

self.city = city

self.state = state

self.zipcode = zipcode

student1 = student("Jane Doe", 25,777, 3.8, "", "", "", "")

print(" ")

print("Student Name: ", student1.name)

print("Student Age: ", student1.age)

print("Student ID: ", student1.studentid)

print("Student GPA: ", student1.gpa)

print("Student Address: ", student1.address)

print("Student City: ", student1.city)

print("Student State: ", student1.state)

print("Student Zip Code: ", student1.zipcode)

print(" ")

teacher1 = teacher("Ms. Cantor", 45, 7, "Python", "", "", "", "")

print("Teacher Name: ", teacher1.name)

print("Teacher Age: ", teacher1.age)

print("Teacher ID: ", teacher1.teacherid)

print("Teacher Course: ", teacher1.subject)

print("Teacher Address: ", teacher1.address)

print("Teacher City: ", teacher1.city)

print("Teacher State: ", teacher1.state)

print("Teacher Zip Code: ", teacher1.zipcode)

**Project #5** (This is another example of two classes and inheritance, the car\_demo class will inherit from the Automobiles class), the \_\_init\_\_ function is called automatically every time the class is used to create an object.

Create a new class and name its **vehicles** and type in the following code below:

Graphical user interface, text, application

Description automatically generated with medium confidenceText

Description automatically generated

Create a second class and label it as **car\_demo**. This will act as the superclass or the main class that will inherit from the Automobile class.

Text

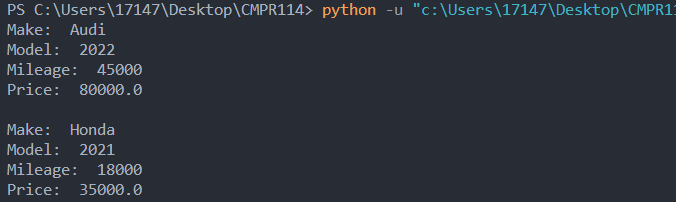
Description automatically generated

**Challenge Exercise #3:** Continuing from project #5, print out another car’s description. See the example below:

Text

Description automatically generated

**#3 print screen the output with the code below here (Need code for each class)**

****

Code:

<From file vehicles>

class automobiles:

def \_\_init\_\_(self, make, model, mileage, price):

self.\_\_make = make

self.\_\_model = model

self.\_\_mileage = mileage

self.\_\_price = price

#mutator methods

def set\_make(self, value):

self.\_\_make = value

def set\_make(self, value):

self.\_\_model = value

def set\_make(self, value):

self.\_\_mileage = value

def set\_make(self, value):

self.\_\_price = value

#assessor methods

def get\_make(self):

return self.\_\_make

def get\_model(self):

return self.\_\_model

def get\_mileage(self):

return self.\_\_mileage

def get\_price(self):

return self.\_\_price

class car\_demo(automobiles):

def \_\_init\_\_(self, make, model, mileage, price):

super().\_\_init\_\_(make, model, mileage, price)

import vehicles

def main():

used\_car1 = vehicles.automobiles("Audi", 2022, 45000, 80000.0)

print("Make: ", used\_car1.get\_make())

print("Model: ", used\_car1.get\_model())

print("Mileage: ", used\_car1.get\_mileage())

print("Price: ", used\_car1.get\_price())

print(" ")

used\_car2 = vehicles.automobiles("Honda", 2021, 18000, 35000.0)

print("Make: ", used\_car2.get\_make())

print("Model: ", used\_car2.get\_model())

print("Mileage: ", used\_car2.get\_mileage())

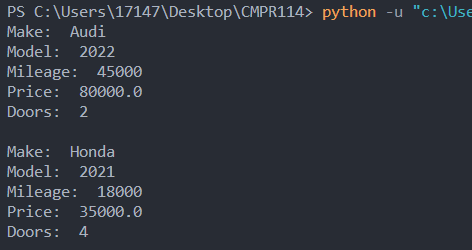
print("Price: ", used\_car2.get\_price())

print(" ")

main()

**Challenge Exercise #4:** now, add the number of doors for each car, be sure to add a mutator and an accessor for the door description.

**#4 Print screen the output with the code below here (Need code for each class)**

****

Code

<From file vehicles>

class automobiles:

def \_\_init\_\_(self, make, model, mileage, price, doors):

self.\_\_make = make

self.\_\_model = model

self.\_\_mileage = mileage

self.\_\_price = price

self.\_\_doors = doors

#mutator methods

def set\_make(self, value):

self.\_\_make = value

def set\_make(self, value):

self.\_\_model = value

def set\_make(self, value):

self.\_\_mileage = value

def set\_make(self, value):

self.\_\_price = value

def set\_doors(self, value):

self.\_\_doors = value

#assessor methods

def get\_make(self):

return self.\_\_make

def get\_model(self):

return self.\_\_model

def get\_mileage(self):

return self.\_\_mileage

def get\_price(self):

return self.\_\_price

def get\_doors(self):

return self.\_\_doors

class car\_demo(automobiles):

def \_\_init\_\_(self, make, model, mileage, price):

super().\_\_init\_\_(make, model, mileage, price)

import vehicles

def main():

used\_car1 = vehicles.automobiles("Audi", 2022, 45000, 80000.0, 2)

print("Make: ", used\_car1.get\_make())

print("Model: ", used\_car1.get\_model())

print("Mileage: ", used\_car1.get\_mileage())

print("Price: ", used\_car1.get\_price())

print("Doors: ", used\_car1.get\_doors())

print(" ")

used\_car2 = vehicles.automobiles("Honda", 2021, 18000, 35000.0, 4)

print("Make: ", used\_car2.get\_make())

print("Model: ", used\_car2.get\_model())

print("Mileage: ", used\_car2.get\_mileage())

print("Price: ", used\_car2.get\_price())

print("Doors: ", used\_car2.get\_doors())

print(" ")

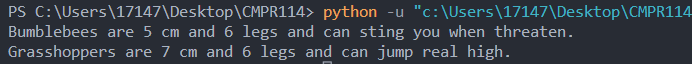
main()

**Challenge Exercise #5:** See the UML diagram below and create the following three classes. The Insect class will be the main or the superclass and the bumblebee and grasshopper classes will act as the two sub-classes. Provide two characteristic behaviors for the bumblebee and grasshopper, be sure to use Mutator and Accessor methods for each class. Print or display the two characteristics of each insect below.

Diagram

Description automatically generated

**#5 Print screen the output with the code below here (Need code for each class)**

****

Code:

class insects:

def \_\_init\_\_(self, size, legs):

self.\_\_size = size

self.\_\_legs = legs

#assessor methods

def get\_size(self):

return self.\_\_size

def get\_legs(self):

return self.\_\_legs

#mutator methods

def set\_size(self, value):

self.\_\_size = value

def set\_legs(self, value):

self.\_\_legs = value

class bumblebees(insects):

def \_\_init\_\_(self, size, legs, defense):

super().\_\_init\_\_(size, legs)

self.\_\_defense = defense

def get\_defense(self):

return self.\_\_defense

def set\_defense(self, value):

self.\_\_defense = value

class grasshoppers(insects):

def \_\_init\_\_(self, size, legs, ability):

super().\_\_init\_\_(size, legs)

self.\_\_ability = ability

def get\_ability(self):

return self.\_\_ability

def set\_defense(self, value):

self.\_\_ability = value

bbee = bumblebees(5, 6, "sting")

print(f"Bumblebees are {bbee.get\_size()} cm and {bbee.get\_legs()} legs and can {bbee.get\_defense()} you when threaten.")

ghop = grasshoppers(7,6, "jump")

print(f"Grasshoppers are {ghop.get\_size()} cm and {ghop.get\_legs()} legs and can {ghop.get\_ability()} real high.")

**Project #6 (Inheritance using TKINTER).**

he super() function is used to call a method from the parent class. In our example, super().\_\_init\_\_() is called within the \_\_init\_\_() method of both EmployeeForm and ManagerForm. This allows the parent class's \_\_init\_\_() method to be executed before the child class's specific code.

Inheritance is a mechanism in object-oriented programming that allows a class to inherit properties and methods from another class. In our code, ManagerForm inherits from EmployeeForm, denoted by class ManagerForm(EmployeeForm):. This means ManagerForm has access to all the attributes and methods defined in EmployeeForm, along with the ability to add or override them as needed.

By calling super().\_\_init\_\_() in the child class's \_\_init\_\_() method, we ensure that the parent class's initialization is performed before any additional customization specific to the child class. This allows us to reuse code and extend the functionality of the parent class in the child class.

A computer screen shot of a program code

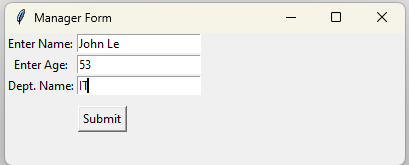
Description automatically generated

A screen shot of a computer program

Description automatically generatedA computer screen with text

Description automatically generated

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



Code:

import tkinter as tk

class EmployeeForm(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Employee Form")

self.geometry("300x200")

self.label\_name = tk.Label(self, text = "Enter Name:")

self.label\_name.grid(row = 0, column = 0)

self.entry\_name = tk.Entry(self)

self.entry\_name.grid(row = 0, column = 1)

self.label\_age = tk.Label(self, text = "Enter Age:")

self.label\_age.grid(row = 1, column = 0)

self.entry\_age = tk.Entry(self)

self.entry\_age.grid(row = 1, column = 1)

self.btnSubmit = tk.Button(self, text = "Submit", command = self.submit)

self.btnSubmit.grid(row = 4, columnspan = 2, pady = 10)

def submit(self):

name = self.entry\_name.get()

age = self.entry\_age.get()

self.display\_employee\_info(name,age)

def display\_employee\_info(self, name, age):

info\_window = tk.Toplevel(self)

info\_window.title("Employee Information")

info\_window.geometry("300x100")

infolbl = tk.Label(info\_window, text = "Emp Info")

infolbl.pack(pady = 10)

namelbl = tk.Label(info\_window, text = f"Name: {name}")

namelbl.pack()

agelbl = tk.Label(info\_window, text = f"Age: {age}")

agelbl.pack()

class ManagerForm(EmployeeForm):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Manager Form")

self.geometry("400x250")

self.deptlbl = tk.Label(self, text = "Dept. Name:")

self.deptlbl.grid(row = 2, column = 0)

self.deptentry = tk.Entry(self)

self.deptentry.grid(row = 2, column = 1)

def submit(self):

name = self.entry\_name.get()

age = self.entry\_age.get()

dept = self.deptentry.get()

self.display\_employee\_info(name, age, dept)

def display\_employee\_info(self, name, age, dept):

info\_window = tk.Toplevel(self)

info\_window.title("Manager Info")

#info\_window.geometry("300x100")

#info\_window.pack(pady = 10)

lblname = tk.Label(info\_window, text = f"Name: {name}")

lblname.pack()

lblage = tk.Label(info\_window, text = f"Age: {age}")

lblage.pack()

lbldept = tk.Label(info\_window, text = f"Dept: {dept}")

lbldept.pack()

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

app = ManagerForm()

app.mainloop()

**Project #7 (Inheritance using TKINTER).**

**A computer screen shot of a program code

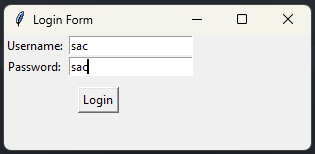
Description automatically generatedA computer screen with colorful text

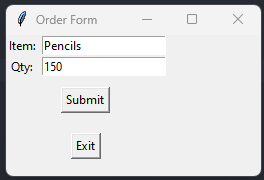
Description automatically generatedA computer screen shot of a program code

Description automatically generatedA computer screen with text

Description automatically generated**

**#7 Print screen the output (for each form) with the code below here.**

****





Code:

import tkinter as tk

class loginform(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Login Form")

self.geometry("200x100")

self.lblusername = tk.Label(self, text="Username: ")

self.lblusername.grid(row=0, column=0)

self.txtusername = tk.Entry(self)

self.txtusername.grid(row=0, column=1)

self.lblpassword = tk.Label(self, text="Password: ")

self.lblpassword.grid(row=1, column=0)

self.txtpassword = tk.Entry(self)

self.txtpassword.grid(row=1, column=1)

self.btnLogin = tk.Button(self, text="Login", command=self.login)

self.btnLogin.grid(row=2, columnspan=2, pady=10)

def login(self):

username = self.txtusername.get()

password = self.txtpassword.get()

if username == "sac" and password == "sac":

self.withdraw()

ordform = orderform(self)

ordform.mainloop()

else:

error\_win = tk.Toplevel()

error\_win.title("Login Error")

lblError = tk.Label(error\_win, text="Invalid username or password.")

lblError.pack(pady=10)

class orderform(tk.Tk):

def \_\_init\_\_(self, loginform):

super().\_\_init\_\_()

self.title("Order Form")

self.login\_form = loginform

self.lblitem = tk.Label(self, text="Item: ")

self.lblitem.grid(row=0, column=0)

self.txtitem = tk.Entry(self)

self.txtitem.grid(row=0, column=1)

self.lblqty = tk.Label(self, text="Qty: ")

self.lblqty.grid(row=1, column=0)

self.txtqty = tk.Entry(self)

self.txtqty.grid(row=1, column=1)

self.btnSubmit = tk.Button(self, text="Submit", command=self.submit)

self.btnSubmit.grid(row=2, columnspan=2, pady=10)

self.btnExit = tk.Button(self, text="Exit", command=self.exit)

self.btnExit.grid(row=3, columnspan=2, pady=10)

def submit(self):

item = self.txtitem.get()

qty = self.txtqty.get()

print(f"Order placed: Item={item}, Quantity={qty}")

def exit(self):

self.withdraw()

self.login\_form.deiconify()

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

login\_form = loginform()

login\_form.mainloop()

**Submit this document to the Module 6 class exercise.**