**Assignments, W1D4**

Algorithmic Coding:

Problem 1: <https://leetcode.com/problems/unique-binary-search-trees-ii/> - Recursion, BST

Problem 2: <https://leetcode.com/problems/validate-binary-search-tree/> Binary Tree, Recursion

Problem 3: <https://leetcode.com/problems/search-a-2d-matrix-ii/>- Nested Arrays, Recursion

Object Oriented Programming:

Problem 1: Battery Upgrade

Use the starter code below for this section. Add a method to the Battery class called upgrade\_battery(). This method should check the battery size and set the capacity to 100 if it isn’t already. Make an electric car with a default battery size, call get\_range() once, and then call get\_range() a second time after upgrading the battery. You should see an increase in the car’s range.

class Car():

"""A simple attempt to represent a car."""

def \_\_init\_\_(self, manufacturer, model, year):

"""Initialize attributes to describe a car."""

self.manufacturer = manufacturer

self.model = model

self.year = year

self.odometer\_reading = 0

def get\_descriptive\_name(self):

"""Return a neatly formatted descriptive name."""

long\_name = f"{self.year} {self.manufacturer} {self.model}"

return long\_name.title()

def read\_odometer(self):

"""Print a statement showing the car's mileage."""

print(f"This car has {self.odometer\_reading} miles on it.")

def update\_odometer(self, mileage):

"""

Set the odometer reading to the given value.

Reject the change if it attempts to roll the odometer back.

"""

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odometer(self, miles):

"""Add the given amount to the odometer reading."""

self.odometer\_reading += miles

class Battery():

"""A simple attempt to model a battery for an electric car."""

def \_\_init\_\_(self, battery\_size=75):

"""Initialize the batteery's attributes."""

self.battery\_size = battery\_size

def describe\_battery(self):

"""Print a statement describing the battery size."""

print(f"This car has a {self.battery\_size}-kWh battery.")

def get\_range(self):

"""Print a statement about the range this battery provides."""

if self.battery\_size == 75:

range = 260

elif self.battery\_size == 100:

range = 315

message = f"This car can go approximately {range}"

message += " miles on a full charge."

print(message)

def upgrade\_battery(self):

“””YOUR CODE STARTS HERE”””

################

“””YOUR CODE ENDS HERE”””

class ElectricCar(Car):

"""Models aspects of a car, specific to electric vehicles."""

def \_\_init\_\_(self, manufacturer, model, year):

"""

Initialize attributes of the parent class.

Then initialize attributes specific to an electric car.

"""

super().\_\_init\_\_(manufacturer, model, year)

self.battery = Battery()

print("Make an electric car, and check the range:")

###YOUR CODE STARTS HERE###

###YOUR CODE ENDS HERE###

print("\nUpgrade the battery, and check the range again:")

###YOUR CODE STARTS HERE###

###YOUR CODE ENDS HERE###

Problem 2: Imported Restaurant

Using your latest Restaurant class, store it in a module. Make a separate file that imports Restaurant. Make a Restaurant instance, and call one of Restaurant’s methods to show that the import statement is working properly.

Problem 3: Imported Admin

Start with the code below, which stores the classes User, Privileges and Admin in one module. Create a separate file, make an Admin instance, and call show\_priveleges() to show that everything is working correctly.

"""A collection of classes for modeling users."""

class User():

"""Represent a simple user profile."""

def \_\_init\_\_(self, first\_name, last\_name, username, email, location):

"""Initialize the user."""

self.first\_name = first\_name.title()

self.last\_name = last\_name.title()

self.username = username

self.email = email

self.location = location.title()

self.login\_attempts = 0

def describe\_user(self):

"""Display a summary of the user's information."""

print(f"\n{self.first\_name} {self.last\_name}")

print(f" Username: {self.username}")

print(f" Email: {self.email}")

print(f" Location: {self.location}")

def greet\_user(self):

"""Display a personalized greeting to the user."""

print(f"\nWelcome back, {self.username}!")

def increment\_login\_attempts(self):

"""Increment the value of login\_attempts."""

self.login\_attempts += 1

def reset\_login\_attempts(self):

"""Reset login\_attempts to 0."""

self.login\_attempts = 0

class Admin(User):

"""A user with administrative privileges."""

def \_\_init\_\_(self, first\_name, last\_name, username, email, location):

"""Initialize the admin."""

super().\_\_init\_\_(first\_name, last\_name, username, email, location)

# Initialize an empty set of privileges.

self.privileges = Privileges()

class Privileges():

"""A class to store an admin's privileges."""

def \_\_init\_\_(self, privileges=[]):

self.privileges = privileges

def show\_privileges(self):

print("\nPrivileges:")

if self.privileges:

for privilege in self.privileges:

print(f"- {privilege}")

else:

print("- This user has no privileges.")