DONE WITHOUT FUNCTIONS

- You are managing employee records for a company. Each employee's record is stored as a tuple containing their name, ID, and department. Create a list of tuples to store these records. Write a function to add a new employee, another to find an employee by their ID, and a function to update the department of a specific employee.

```
t1=("Mehak",1234,"cse")
t2=("b",2345,"ece")
t3=("c",3456,"mech")
l1=[t1,t2,t3]
print(l1)
l1.append(("d",12345,"sdf"))
print(l1)
for i in range(0,len(l1)):
   if l1[i][1]==1234:
    print("Employee idex is:",i)
    print(l1[i])
temp=list(l1[2])
temp[2]="bcd"
l1[2]=tuple(temp);
print(l1)
```

```
[('Mehak', 1234, 'cse'), ('b', 2345, 'ece'), ('c', 3456, 'mech')]
[('Mehak', 1234, 'cse'), ('b', 2345, 'ece'), ('c', 3456, 'mech'),
('d', 12345, 'sdf')]
Employee idex is: 0
('Mehak', 1234, 'cse')
[('Mehak', 1234, 'cse'), ('b', 2345, 'ece'), ('c', 3456, 'bcd'), ('d',
12345, 'sdf')]
```

1.
   - You are working on a mapping application that stores geographical coordinates (latitude and longitude) as tuples. Create a list of these coordinate tuples. Write functions to add a new coordinate, find a specific coordinate, and calculate the distance between two coordinates.

```
from math import radians, sin, cos, sqrt, atan2
coord1 = (40.7128, -74.0060)  # New York City
coord2 = (34.0522, -118.2437) # Los Angeles
coord3 = (51.5074, -0.1278)   # London
coordinates = [coord1, coord2, coord3]
print("Initial list of coordinates:", coordinates)
coordinates.append((35.6895, 139.6917))  # Tokyo
print("List of coordinates after adding a new one:", coordinates)
```

```
# Find a specific coordinate (e.g., Los Angeles)
target = (34.0522, -118.2437)
for i in range(len(coordinates)):
    if coordinates[i] == target:
        print("Coordinate found at index:", i)
        print("Found coordinate:", coordinates[i])

# Calculate the distance between two coordinates using the Haversine
formula
coord1 = coordinates[0]  # New York City
coord2 = coordinates[1]  # Los Angeles

# Convert latitude and longitude from degrees to radians
lat1, lon1 = radians(coord1[0]), radians(coord1[1])
lat2, lon2 = radians(coord2[0]), radians(coord2[1])

# Haversine formula
dlon = lon2 - lon1
dlat = lat2 - lat1
a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
c = 2 * atan2(sqrt(a), sqrt(1 - a))

# Radius of Earth in kilometers (mean radius)
R = 6371.01
distance = R * c

print("Distance between New York City and Los Angeles:", distance,
"km")

# Modify the coordinates of London (index 2)
temp = list(coordinates[2])
temp[0], temp[1] = 48.8566, 2.3522  # Change coordinates to Paris
coordinates[2] = tuple(temp)
print("List of coordinates after modification:", coordinates)
```

```
Initial list of coordinates: [(40.7128, -74.006), (34.0522, -
118.2437), (51.5074, -0.1278)]
List of coordinates after adding a new one: [(40.7128, -74.006),
(34.0522, -118.2437), (51.5074, -0.1278), (35.6895, 139.6917)]
Coordinate found at index: 1
Found coordinate: (34.0522, -118.2437)
Distance between New York City and Los Angeles: 3935.7524322054765 km
List of coordinates after modification: [(40.7128, -74.006), (34.0522,
-118.2437), (48.8566, 2.3522), (35.6895, 139.6917)]
```

- You have a list of students, where each student is represented as a tuple containing their name, age, and grade. Write a function to add a new student, another function to remove a student by name, and a function to find the student with the highest grade.

```
student1 = ("Alice", 20, 88)
student2 = ("Bob", 22, 76)
student3 = ("Charlie", 21, 95)
students = [student1, student2, student3]
print("Initial list of students:", students)
new_student = ("David", 23, 82)
students.append(new_student)
print("List of students after adding a new one:", students)

for i in range(len(students)):
    if students[i][0] == "Bob":
        del students[i]
        break
print("List of students after removing Bob:", students)
highest_grade = -1
for student in students:
    if student[2] > highest_grade:
        highest_grade = student[2]
        top_student = student
print("Highest grade:",highest_grade)
```

```
Initial list of students: [('Alice', 20, 88), ('Bob', 22, 76),
('Charlie', 21, 95)]
List of students after adding a new one: [('Alice', 20, 88), ('Bob',
22, 76), ('Charlie', 21, 95), ('David', 23, 82)]
List of students after removing Bob: [('Alice', 20, 88), ('Charlie',
21, 95), ('David', 23, 82)]
Highest grade: 95
```

4. on system for a software application where configurations are stored as tuples. Each configuration contains a setting name and its value. Write a function to add a new configuration, check if a specific setting exists, and retrieve the value of a specific setting.

```
config1 = ("volume", 75)
config2 = ("brightness", 50)
config3 = ("resolution", "1080p")
configurations = [config1, config2, config3]

# Print initial list of configurations
print("Initial list of configurations:", configurations)

# Adding a new configuration
new_config = ("language", "English")
configurations.append(new_config)
print("List of configurations after adding a new one:",
configurations)

setting_to_check = "brightness"
for i in range(len(configurations)):
```

```
    if configurations[i][0] == setting_to_check:
        print("Setting exists at index:", i)
        print("Value of the setting:", configurations[i][1])
setting_to_retrieve = "volume"
for config in configurations:
    if config[0] == setting_to_retrieve:
        print("Value of the setting:", config[1])
```

```
Initial list of configurations: [('volume', 75), ('brightness', 50),
('resolution', '1080p')]
List of configurations after adding a new one: [('volume', 75),
('brightness', 50), ('resolution', '1080p'), ('language', 'English')]
Setting exists at index: 1
Value of the setting: 50
Value of the setting: 75
```

- Develop a contact list application where each contact is stored as a tuple containing their name, phone number, and email address. Write functions to add a new contact, search for a contact by name, and update the phone number of a specific contact.

```
contact1 = ("Alice", "123-456-7890", "alice@example.com")
contact2 = ("Bob", "234-567-8901", "bob@example.com")
contact3 = ("Charlie", "345-678-9012", "charlie@example.com")
contacts = [contact1, contact2, contact3]

# Print initial list of contacts
print("Initial list of contacts:", contacts)

# Adding a new contact
new_contact = ("David", "456-789-0123", "david@example.com")
contacts.append(new_contact)
print("List of contacts after adding a new one:", contacts)

# Search for a contact by name
name_to_search = "Bob"
for contact in contacts:
    if contact[0] == name_to_search:
        print("Contact found:", contact)
temp=list(contacts[1]);
temp[1]="+91-xxxxxxxxxx"
contacts[1]=tuple(temp)
print("List of contacts after updating the phone number:", contacts)
```

```
Initial list of contacts: [('Alice', '123-456-7890',
'alice@example.com'), ('Bob', '234-567-8901', 'bob@example.com'),
('Charlie', '345-678-9012', 'charlie@example.com')]
List of contacts after adding a new one: [('Alice', '123-456-7890',
'alice@example.com'), ('Bob', '234-567-8901', 'bob@example.com'),
('Charlie', '345-678-9012', 'charlie@example.com'), ('David', '456-
789-0123', 'david@example.com')]
```

```
Contact found: ('Bob', '234-567-8901', 'bob@example.com')
List of contacts after updating the phone number: [('Alice', '123-456-
7890', 'alice@example.com'), ('Bob', '+91-xxxxxxxxxx',
'bob@example.com'), ('Charlie', '345-678-9012',
'charlie@example.com'), ('David', '456-789-0123',
'david@example.com')]
```

- Represent points in 3D space as tuples containing x, y, and z coordinates. Create a list of these point tuples. Write functions to add a new point, calculate the distance between two points, and find the point closest to the origin.

```
import math
# Initial list of points in 3D space as tuples (x, y, z)
point1 = (1, 2, 3)
point2 = (4, 5, 6)
point3 = (7, 8, 9)
points = [point1, point2, point3]

# Print initial list of points
print("Initial list of points:", points)

# Adding a new point
new_point = (10, 11, 12)
points.append(new_point)
print("List of points after adding a new one:", points)
pointA = points[0]  # (1, 2, 3)
pointB = points[1]  # (4, 5, 6)

distance = math.sqrt((pointA[0] - pointB[0])**2 + (pointA[1] -
pointB[1])**2 + (pointA[2] - pointB[2])**2)
print(f"Distance between points {pointA} and {pointB}: {distance}")
origin=(0,0,0)
for point in points:
  distance=min(math.sqrt((point[0]-origin[0])**2+(point[1]-
origin[1])**2+(point[2]-origin[2])**2),distance)
print("Min distance is: ",distance)


Initial list of points: [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
List of points after adding a new one: [(1, 2, 3), (4, 5, 6), (7, 8,
9), (10, 11, 12)]
Distance between points (1, 2, 3) and (4, 5, 6): 5.196152422706632
Min distance is:  3.7416573867739413
```

- Create a restaurant menu where each item is represented as a tuple containing the item name, description, and price. Write functions to add a new menu item, update the description of an existing item, and find all items below a certain price.

```
item1 = ("Burger", "A delicious beef burger with lettuce, tomato, and
```

```
cheese.", 8.99)
item2 = ("Pizza", "A large pizza with your choice of toppings.",
12.99)
item3 = ("Salad", "A fresh garden salad with a variety of
vegetables.", 7.49)
menu = [item1, item2, item3]

# Print initial list of menu items
print("Initial menu items:", menu)

# Adding a new menu item
new_item = ("Pasta", "A classic Italian pasta with marinara sauce.",
10.99)
menu.append(new_item)
print("Menu after adding a new item:", menu)
item_name_to_update = "Pizza"
new_description = "A large pizza with your choice of toppings and
extra cheese."
for i in range(len(menu)):
    if menu[i][0] == item_name_to_update:
        menu[i] = (menu[i][0], new_description, menu[i][2])
        break
print("Menu after updating the description:", menu)
price_limit = 9.00
for i in menu:
  if i[2]<price_limit:
    print("Afordable items are:",i)
```

```
Initial menu items: [('Burger', 'A delicious beef burger with lettuce,
tomato, and cheese.', 8.99), ('Pizza', 'A large pizza with your choice
of toppings.', 12.99), ('Salad', 'A fresh garden salad with a variety
of vegetables.', 7.49)]
Menu after adding a new item: [('Burger', 'A delicious beef burger
with lettuce, tomato, and cheese.', 8.99), ('Pizza', 'A large pizza
with your choice of toppings.', 12.99), ('Salad', 'A fresh garden
salad with a variety of vegetables.', 7.49), ('Pasta', 'A classic
Italian pasta with marinara sauce.', 10.99)]
Menu after updating the description: [('Burger', 'A delicious beef
burger with lettuce, tomato, and cheese.', 8.99), ('Pizza', 'A large
pizza with your choice of toppings and extra cheese.', 12.99),
('Salad', 'A fresh garden salad with a variety of vegetables.', 7.49),
('Pasta', 'A classic Italian pasta with marinara sauce.', 10.99)]
Afordable items are: ('Burger', 'A delicious beef burger with lettuce,
tomato, and cheese.', 8.99)
Afordable items are: ('Salad', 'A fresh garden salad with a variety of
vegetables.', 7.49)
```

```
#    - Develop an immutable shopping cart system where each cart item
is represented as a tuple containing the item name, quantity, and
price. Write functions to add a new item to the cart, update the
```

```
quantity of an existing item, and calculate the total cost of the
cart.

item1 = ("Apple", 3, 0.99)
item2 = ("Banana", 6, 0.59)
item3 = ("Orange", 2, 1.29)
cart = [item1, item2, item3]

# Print initial cart items
print("Initial cart items:", cart)

# Function to add a new item to the cart
new_item = ("Grapes", 4, 2.99)
cart = cart + [new_item]  # Immutable addition
print("Cart after adding a new item:", cart)
item_name_to_update = "Banana"
new_quantity = 10
for i in cart:
    i[0] == item_name_to_update
    i = (i[0], new_quantity, i[2])
    break
print("Cart after updating the quantity:", cart)
total_cost = 0
for item in cart:
    total_cost += item[1] * item[2]
print("Total cost of the cart:", total_cost)
```

```
Initial cart items: [('Apple', 3, 0.99), ('Banana', 6, 0.59),
('Orange', 2, 1.29)]
Cart after adding a new item: [('Apple', 3, 0.99), ('Banana', 6,
0.59), ('Orange', 2, 1.29), ('Grapes', 4, 2.99)]
Cart after updating the quantity: [('Apple', 3, 0.99), ('Banana', 6,
0.59), ('Orange', 2, 1.29), ('Grapes', 4, 2.99)]
Total cost of the cart: 21.05
```