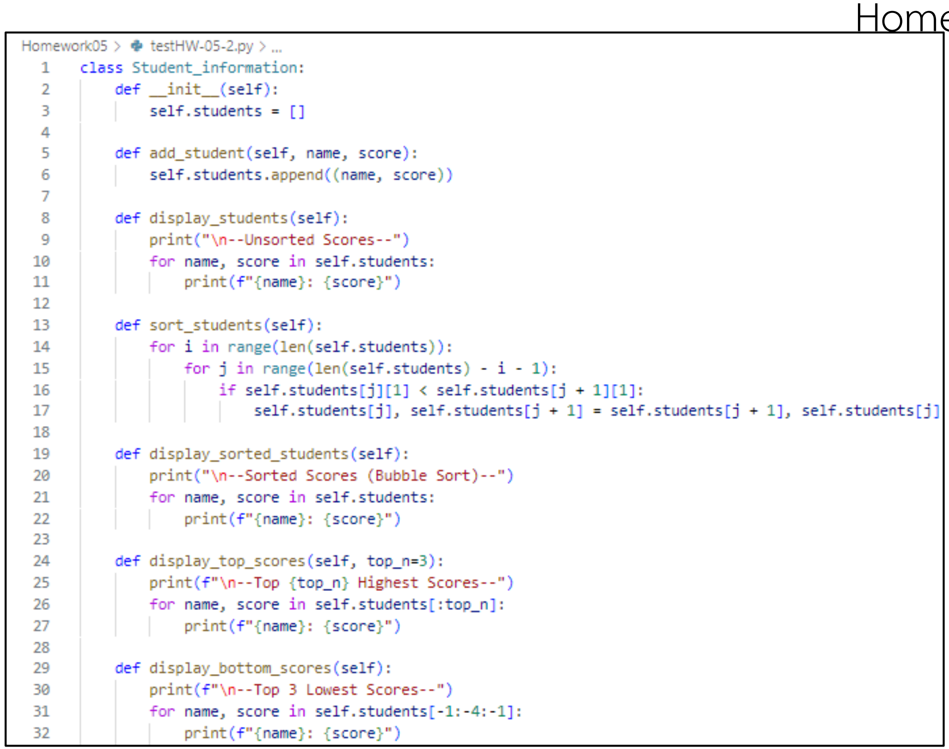
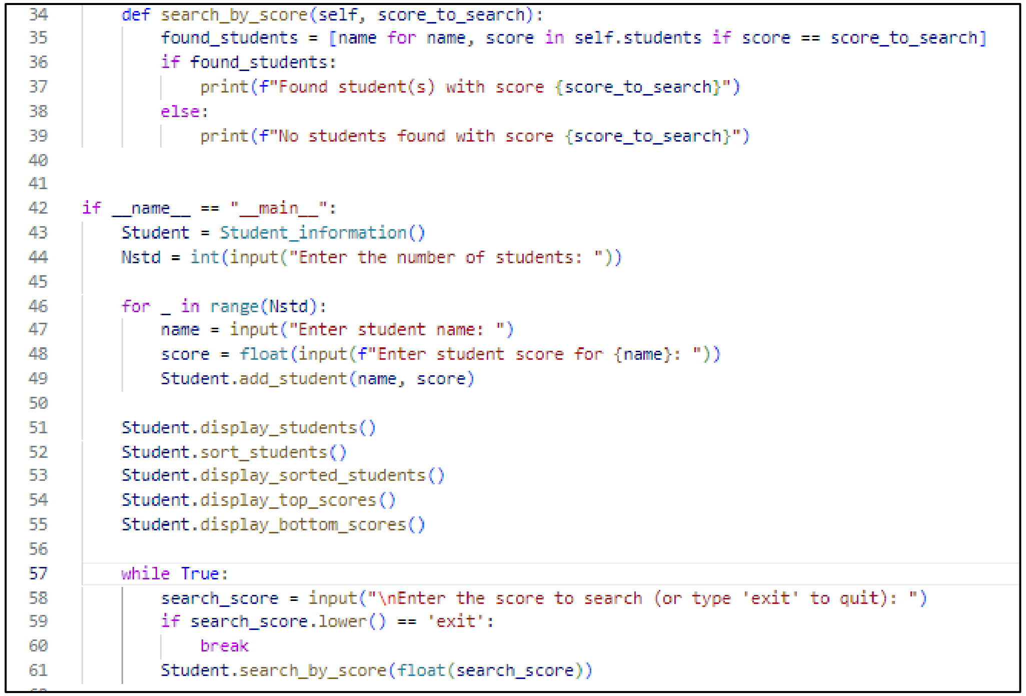
Homework 5

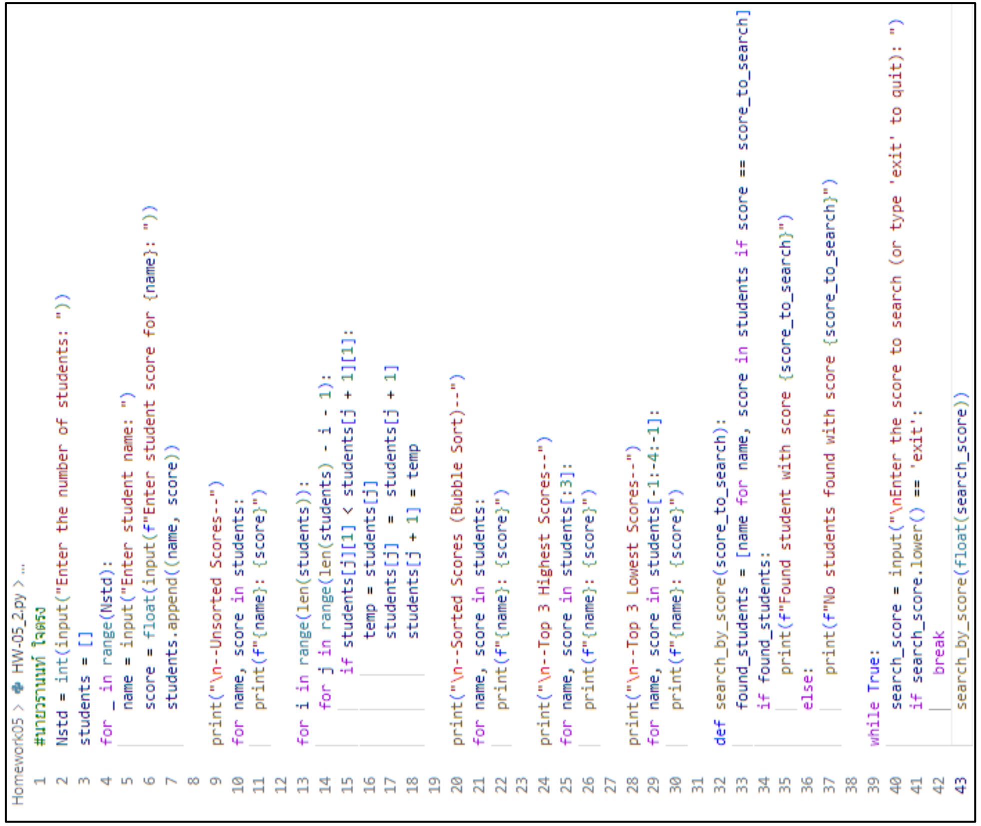


Homework05 > testHW-05-2.py > ...

```python
class Student_information:
    def __init__(self):
        self.students = []

    def add_student(self, name, score):
        self.students.append((name, score))

    def display_students(self):
        print("\n--Unsorted Scores--")
        for name, score in self.students:
            print(f"{name}: {score}")

    def sort_students(self):
        for i in range(len(self.students)):
            for j in range(len(self.students) - i - 1):
                if self.students[j][1] < self.students[j + 1][1]:
                    self.students[j], self.students[j + 1] = self.students[j + 1], self.students[j]

    def display_sorted_students(self):
        print("\n--Sorted Scores (Bubble Sort)--")
        for name, score in self.students:
            print(f"{name}: {score}")

    def display_top_scores(self, top_n=3):
        print(f"\n--Top {top_n} Highest Scores--")
        for name, score in self.students[:top_n]:
            print(f"{name}: {score}")

    def display_bottom_scores(self):
        print(f"\n--Top 3 Lowest Scores--")
        for name, score in self.students[-1:-4:-1]:
            print(f"{name}: {score}")
```



```python
    def search_by_score(self, score_to_search):
        found_students = [name for name, score in self.students if score == score_to_search]
        if found_students:
            print(f"Found student(s) with score {score_to_search}")
        else:
            print(f"No students found with score {score_to_search}")


if __name__ == "__main__":
    Student = Student_information()
    Nstd = int(input("Enter the number of students: "))

    for _ in range(Nstd):
        name = input("Enter student name: ")
        score = float(input(f"Enter student score for {name}: "))
        Student.add_student(name, score)

    Student.display_students()
    Student.sort_students()
    Student.display_sorted_students()
    Student.display_top_scores()
    Student.display_bottom_scores()

    while True:
        search_score = input("\nEnter the score to search (or type 'exit' to quit): ")
        if search_score.lower() == 'exit':
            break
        Student.search_by_score(float(search_score))
```

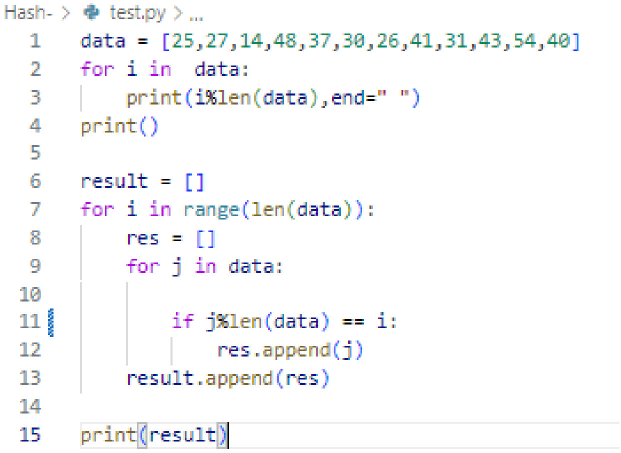


Homework05 > HW-05_2.py > ...

```python
#นายวรานนท์ ใจตรง
Nstd = int(input("Enter the number of students: "))
students = []
for _ in range(Nstd):
    name = input("Enter student name: ")
    score = float(input(f"Enter student score for {name}: "))
    students.append((name, score))

print("\n--Unsorted Scores--")
for name, score in students:
    print(f"{name}: {score}")

for i in range(len(students)):
    for j in range(len(students) - i - 1):
        if students[j][1] < students[j + 1][1]:
            temp = students[j]
            students[j]  =  students[j + 1]
            students[j + 1] = temp

print("\n--Sorted Scores (Bubble Sort)--")
for name, score in students:
    print(f"{name}: {score}")

print("\n--Top 3 Highest Scores--")
for name, score in students[:3]:
    print(f"{name}: {score}")

print("\n--Top 3 Lowest Scores--")
for name, score in students[-1:-4:-1]:
    print(f"{name}: {score}")

def search_by_score(score_to_search):
    found_students = [name for name, score in students if score == score_to_search]
    if found_students:
        print(f"Found student with score {score_to_search}")
    else:
        print(f"No students found with score {score_to_search}")

while True:
    search_score = input("\nEnter the score to search (or type 'exit' to quit): ")
    if search_score.lower() == 'exit':
        break
    search_by_score(float(search_score))
```

โค้ดช่วย Hash

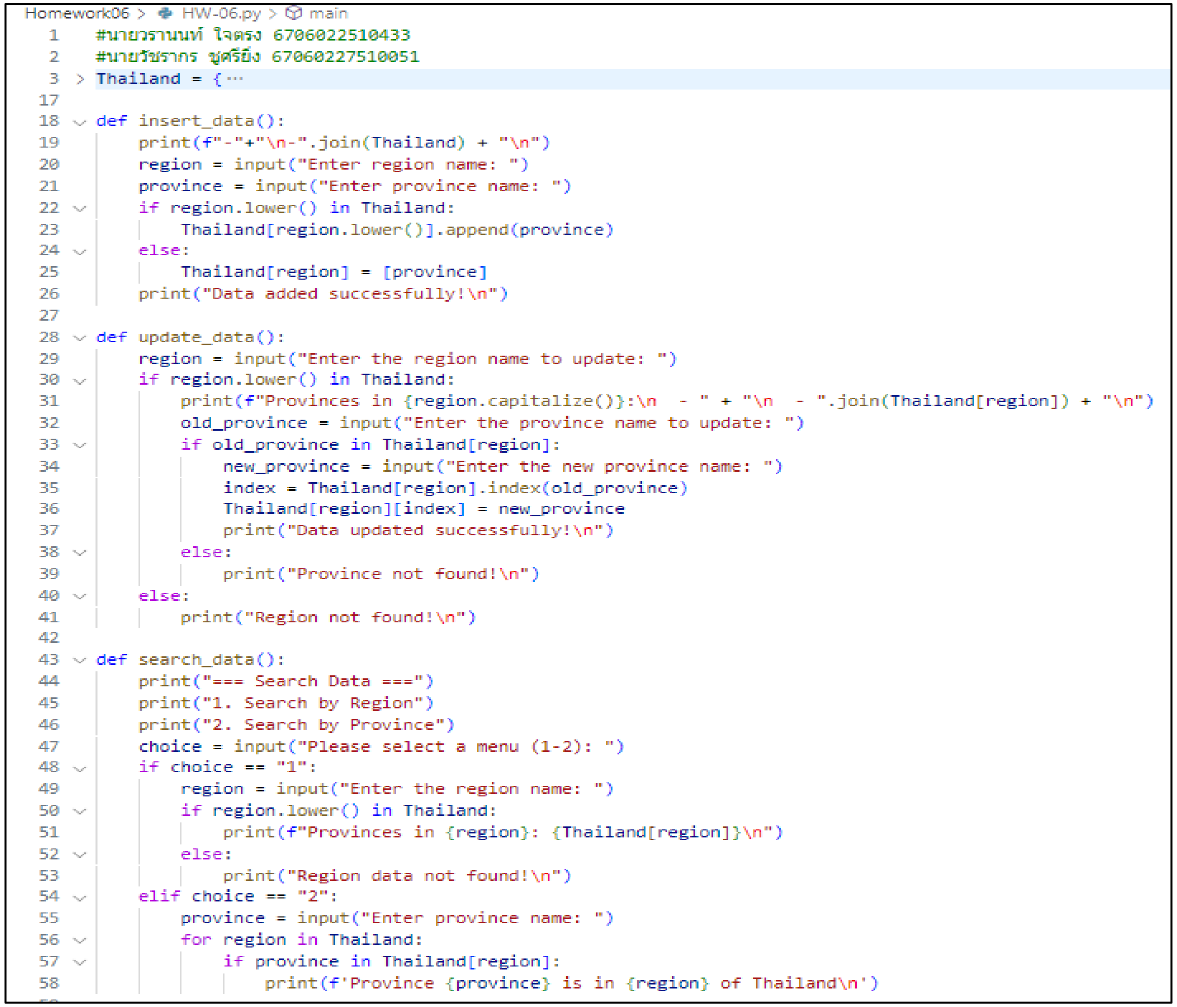
ใช้ mod ข้อมูลทีละตัว



Hash- > test.py > ...

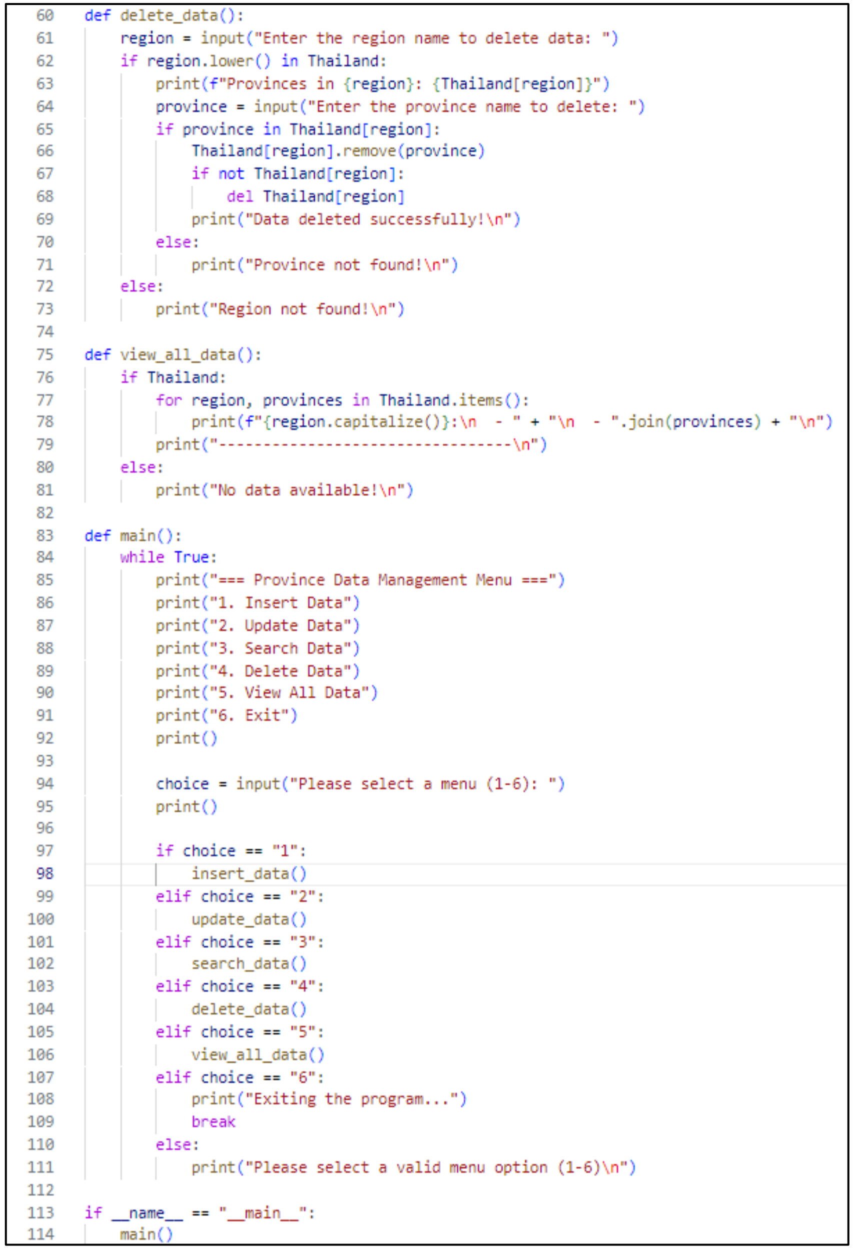
```python
data = [25,27,14,48,37,30,26,41,31,43,54,40]
for i in  data:
    print(i%len(data),end=" ")
print()

result = []
for i in range(len(data)):
    res = []
    for j in data:

        if j%len(data) == i:
            res.append(j)
    result.append(res)

print(result)
```

Homework 6



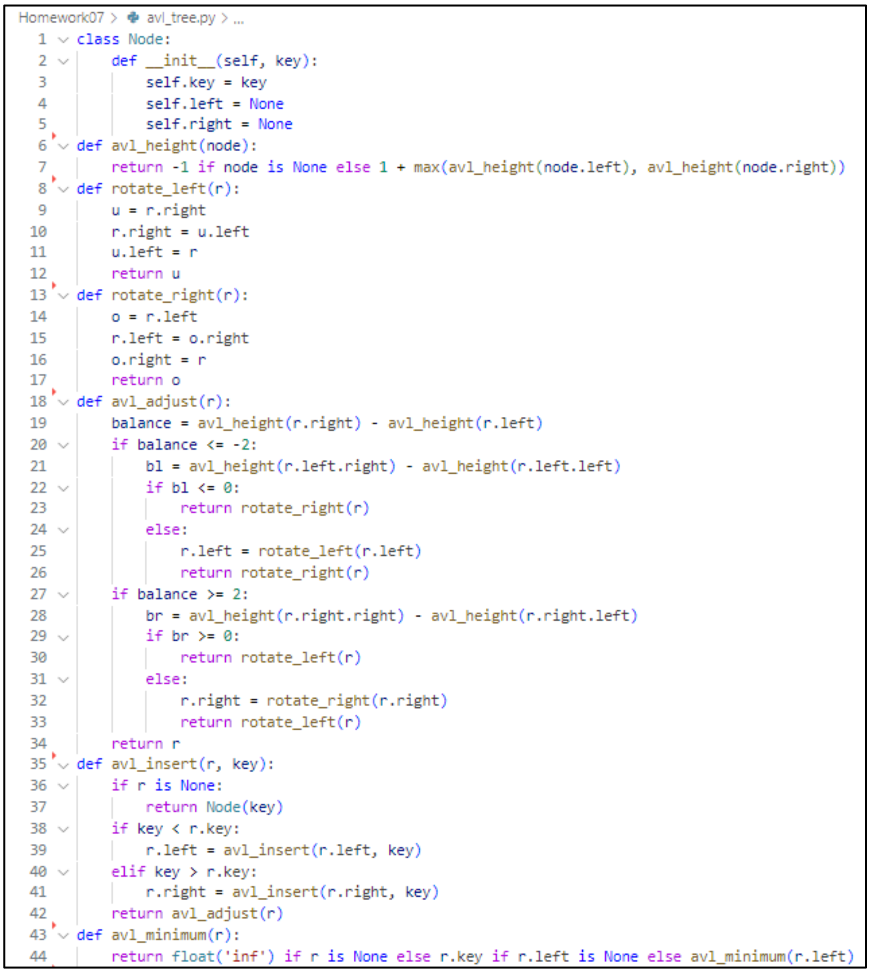
Homework06 > HW-06.py > main

```python
#นายวรานนท์ ใจตรง 6706022510433
#นายวัชรากร ชูศรียิ่ง 67060227510051
Thailand = {···

def insert_data():
    print(f"-"+"\n-".join(Thailand) + "\n")
    region = input("Enter region name: ")
    province = input("Enter province name: ")
    if region.lower() in Thailand:
        Thailand[region.lower()].append(province)
    else:
        Thailand[region] = [province]
    print("Data added successfully!\n")

def update_data():
    region = input("Enter the region name to update: ")
    if region.lower() in Thailand:
        print(f"Provinces in {region.capitalize()}:\n  - " + "\n  - ".join(Thailand[region]) + "\n")
        old_province = input("Enter the province name to update: ")
        if old_province in Thailand[region]:
            new_province = input("Enter the new province name: ")
            index = Thailand[region].index(old_province)
            Thailand[region][index] = new_province
            print("Data updated successfully!\n")
        else:
            print("Province not found!\n")
    else:
        print("Region not found!\n")

def search_data():
    print("=== Search Data ===")
    print("1. Search by Region")
    print("2. Search by Province")
    choice = input("Please select a menu (1-2): ")
    if choice == "1":
        region = input("Enter the region name: ")
        if region.lower() in Thailand:
            print(f"Provinces in {region}: {Thailand[region]}\n")
        else:
            print("Region data not found!\n")
    elif choice == "2":
        province = input("Enter province name: ")
        for region in Thailand:
            if province in Thailand[region]:
                print(f'Province {province} is in {region} of Thailand\n')
```



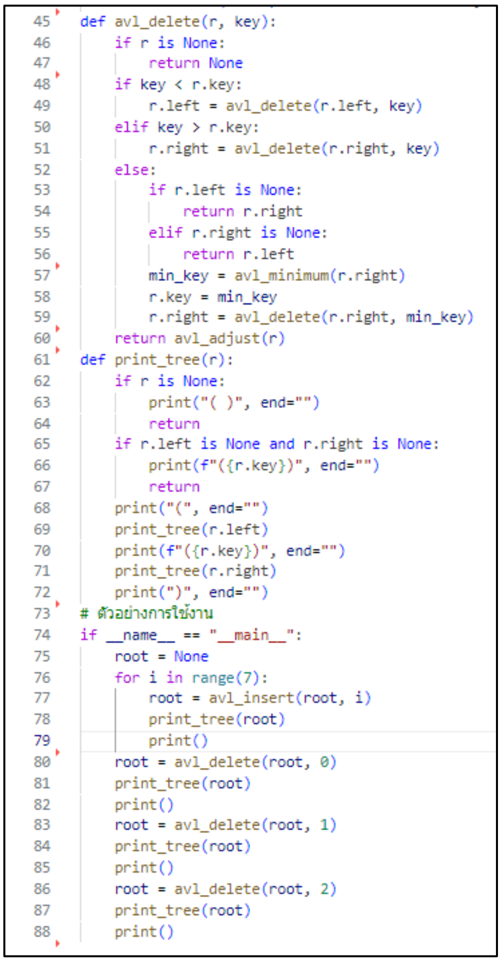
```python
def delete_data():
    region = input("Enter the region name to delete data: ")
    if region.lower() in Thailand:
        print(f"Provinces in {region}: {Thailand[region]}")
        province = input("Enter the province name to delete: ")
        if province in Thailand[region]:
            Thailand[region].remove(province)
            if not Thailand[region]:
                del Thailand[region]
            print("Data deleted successfully!\n")
        else:
            print("Province not found!\n")
    else:
        print("Region not found!\n")

def view_all_data():
    if Thailand:
        for region, provinces in Thailand.items():
            print(f"{region.capitalize()}:\n  - " + "\n  - ".join(provinces) + "\n")
        print("--------------------------------\n")
    else:
        print("No data available!\n")

def main():
    while True:
        print("=== Province Data Management Menu ===")
        print("1. Insert Data")
        print("2. Update Data")
        print("3. Search Data")
        print("4. Delete Data")
        print("5. View All Data")
        print("6. Exit")
        print()

        choice = input("Please select a menu (1-6): ")
        print()

        if choice == "1":
            insert_data()
        elif choice == "2":
            update_data()
        elif choice == "3":
            search_data()
        elif choice == "4":
            delete_data()
        elif choice == "5":
            view_all_data()
        elif choice == "6":
            print("Exiting the program...")
            break
        else:
            print("Please select a valid menu option (1-6)\n")

if __name__ == "__main__":
    main()
```

AVL Tree



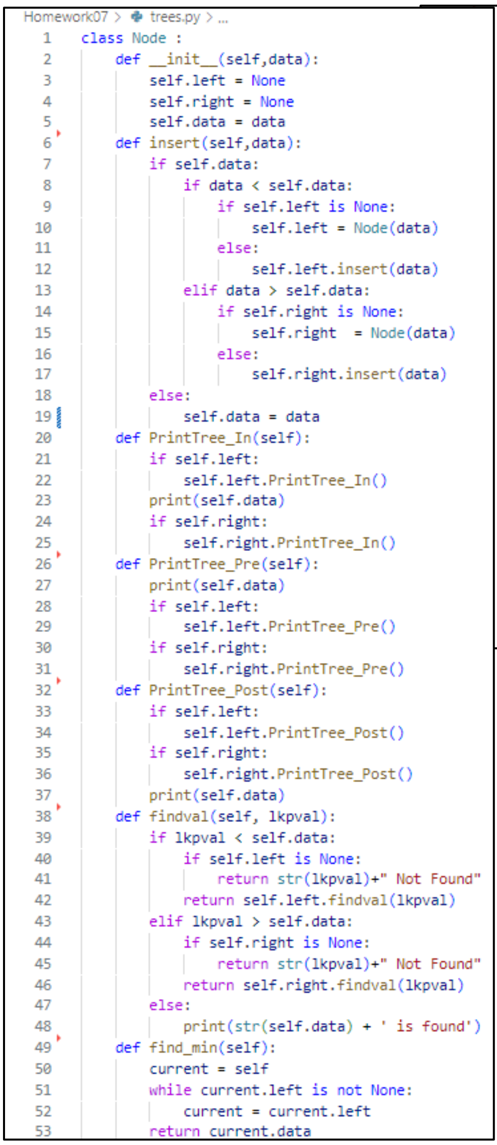
Homework07 > avl_tree.py > ...

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
def avl_height(node):
    return -1 if node is None else 1 + max(avl_height(node.left), avl_height(node.right))
def rotate_left(r):
    u = r.right
    r.right = u.left
    u.left = r
    return u
def rotate_right(r):
    o = r.left
    r.left = o.right
    o.right = r
    return o
def avl_adjust(r):
    balance = avl_height(r.right) - avl_height(r.left)
    if balance <= -2:
        bl = avl_height(r.left.right) - avl_height(r.left.left)
        if bl <= 0:
            return rotate_right(r)
        else:
            r.left = rotate_left(r.left)
            return rotate_right(r)
    if balance >= 2:
        br = avl_height(r.right.right) - avl_height(r.right.left)
        if br >= 0:
            return rotate_left(r)
        else:
            r.right = rotate_right(r.right)
            return rotate_left(r)
    return r
def avl_insert(r, key):
    if r is None:
        return Node(key)
    if key < r.key:
        r.left = avl_insert(r.left, key)
    elif key > r.key:
        r.right = avl_insert(r.right, key)
    return avl_adjust(r)
def avl_minimum(r):
    return float('inf') if r is None else r.key if r.left is None else avl_minimum(r.left)
```



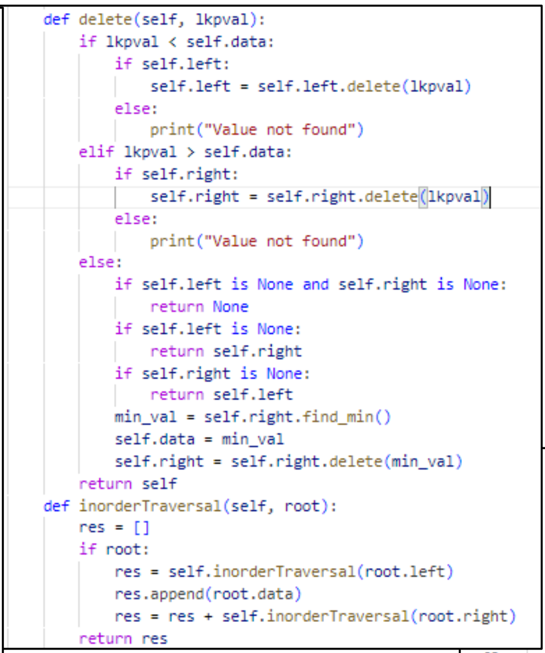
```python
def avl_delete(r, key):
    if r is None:
        return None
    if key < r.key:
        r.left = avl_delete(r.left, key)
    elif key > r.key:
        r.right = avl_delete(r.right, key)
    else:
        if r.left is None:
            return r.right
        elif r.right is None:
            return r.left
        min_key = avl_minimum(r.right)
        r.key = min_key
        r.right = avl_delete(r.right, min_key)
    return avl_adjust(r)
def print_tree(r):
    if r is None:
        print("( )", end="")
        return
    if r.left is None and r.right is None:
        print(f"({r.key})", end="")
        return
    print("(", end="")
    print_tree(r.left)
    print(f"({r.key})", end="")
    print_tree(r.right)
    print(")", end="")
# ตัวอย่างการใช้งาน
if __name__ == "__main__":
    root = None
    for i in range(7):
        root = avl_insert(root, i)
        print_tree(root)
        print()
    root = avl_delete(root, 0)
    print_tree(root)
    print()
    root = avl_delete(root, 1)
    print_tree(root)
    print()
    root = avl_delete(root, 2)
    print_tree(root)
    print()
```

Trees

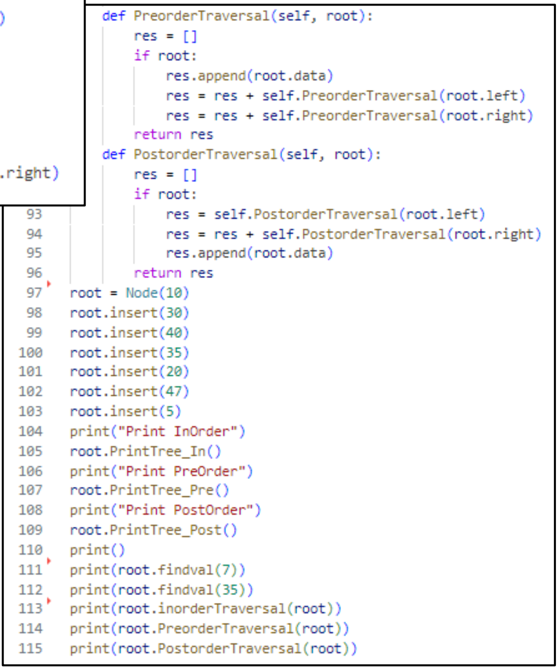


Homework07 > trees.py > ...

```python
class Node :
    def __init__(self,data):
        self.left = None
        self.right = None
        self.data = data
    def insert(self,data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right  = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data
    def PrintTree_In(self):
        if self.left:
            self.left.PrintTree_In()
        print(self.data)
        if self.right:
            self.right.PrintTree_In()
    def PrintTree_Pre(self):
        print(self.data)
        if self.left:
            self.left.PrintTree_Pre()
        if self.right:
            self.right.PrintTree_Pre()
    def PrintTree_Post(self):
        if self.left:
            self.left.PrintTree_Post()
        if self.right:
            self.right.PrintTree_Post()
        print(self.data)
    def findval(self, lkpval):
        if lkpval < self.data:
            if self.left is None:
                return str(lkpval)+" Not Found"
            return self.left.findval(lkpval)
        elif lkpval > self.data:
            if self.right is None:
                return str(lkpval)+" Not Found"
            return self.right.findval(lkpval)
        else:
            print(str(self.data) + ' is found')
    def find_min(self):
        current = self
        while current.left is not None:
            current = current.left
        return current.data
```

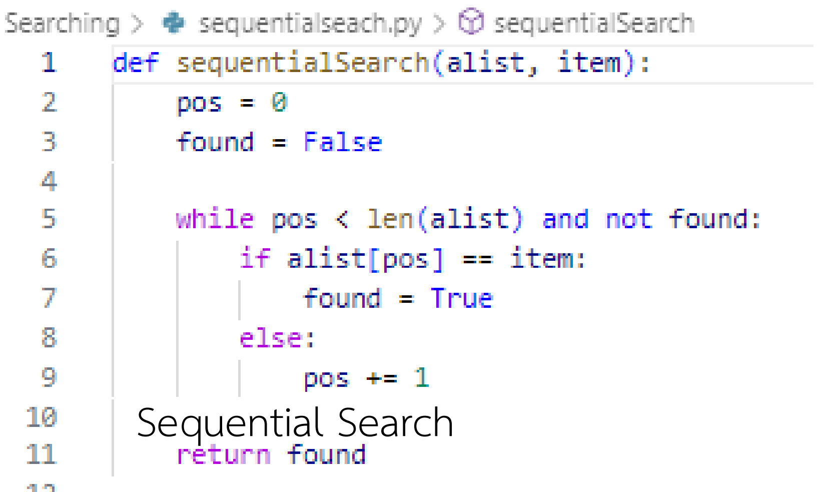


```python
    def delete(self, lkpval):
        if lkpval < self.data:
            if self.left:
                self.left = self.left.delete(lkpval)
            else:
                print("Value not found")
        elif lkpval > self.data:
            if self.right:
                self.right = self.right.delete(lkpval)
            else:
                print("Value not found")
        else:
            if self.left is None and self.right is None:
                return None
            if self.left is None:
                return self.right
            if self.right is None:
                return self.left
            min_val = self.right.find_min()
            self.data = min_val
            self.right = self.right.delete(min_val)
        return self
    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.data)
            res = res + self.inorderTraversal(root.right)
        return res
```



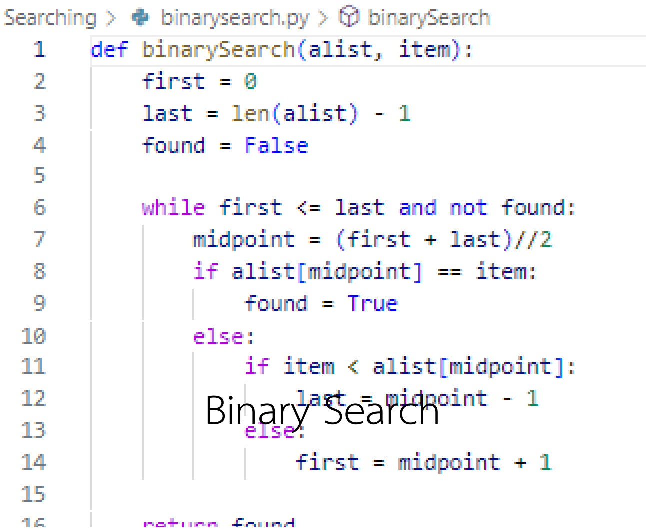
```python
    def PreorderTraversal(self, root):
        res = []
        if root:
            res.append(root.data)
            res = res + self.PreorderTraversal(root.left)
            res = res + self.PreorderTraversal(root.right)
        return res
    def PostorderTraversal(self, root):
        res = []
        if root:
            res = self.PostorderTraversal(root.left)
            res = res + self.PostorderTraversal(root.right)
            res.append(root.data)
        return res
root = Node(10)
root.insert(30)
root.insert(40)
root.insert(35)
root.insert(20)
root.insert(47)
root.insert(5)
print("Print InOrder")
root.PrintTree_In()
print("Print PreOrder")
root.PrintTree_Pre()
print("Print PostOrder")
root.PrintTree_Post()
print()
print(root.findval(7))
print(root.findval(35))
print(root.inorderTraversal(root))
print(root.PreorderTraversal(root))
print(root.PostorderTraversal(root))
```

Search



```python
def sequentialSearch(alist, item):
    pos = 0
    found = False

    while pos < len(alist) and not found:
        if alist[pos] == item:
            found = True
        else:
            pos += 1

    return found
```

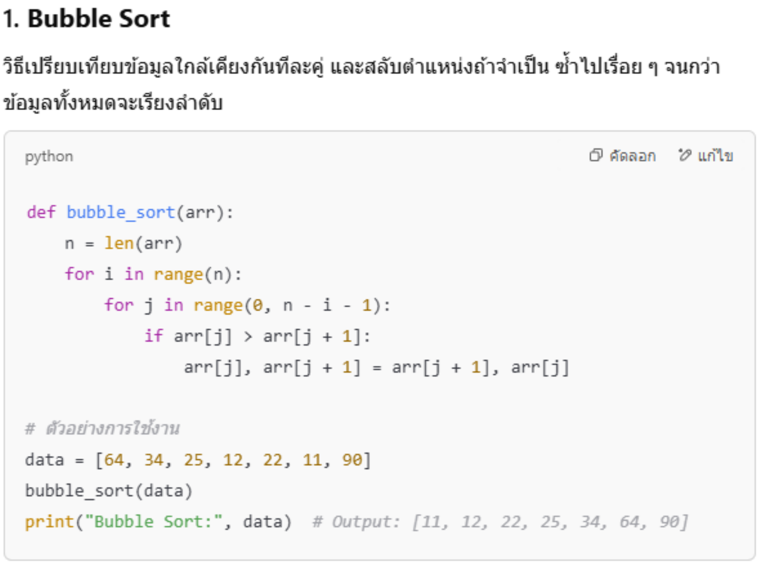
Sequential Search



```python
def binarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False

    while first <= last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1

    return found
```

Binary Search

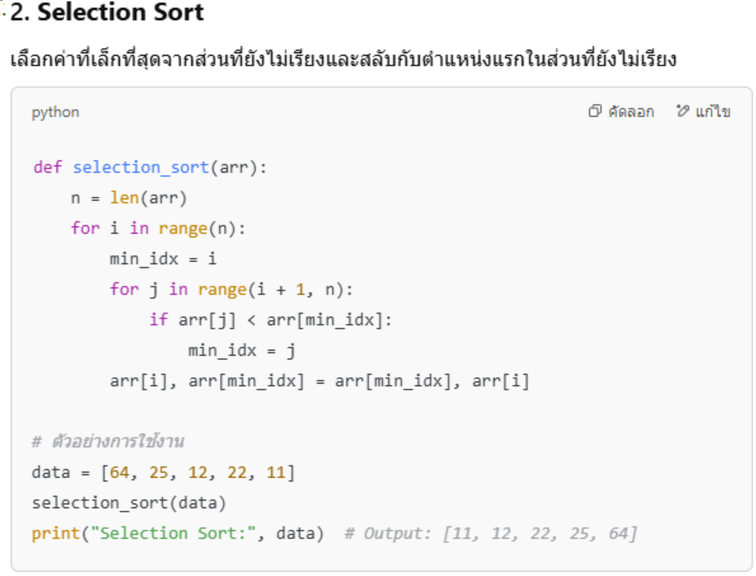
Sorting



1. Bubble Sort

วิธีเปรียบเทียบข้อมูลใกล้เคียงกันทีละคู่ และสลับตำแหน่งถ้าจำเป็น ซ้ำไปเรื่อย ๆ จนกว่าข้อมูลทั้งหมดจะเรียงลำดับ

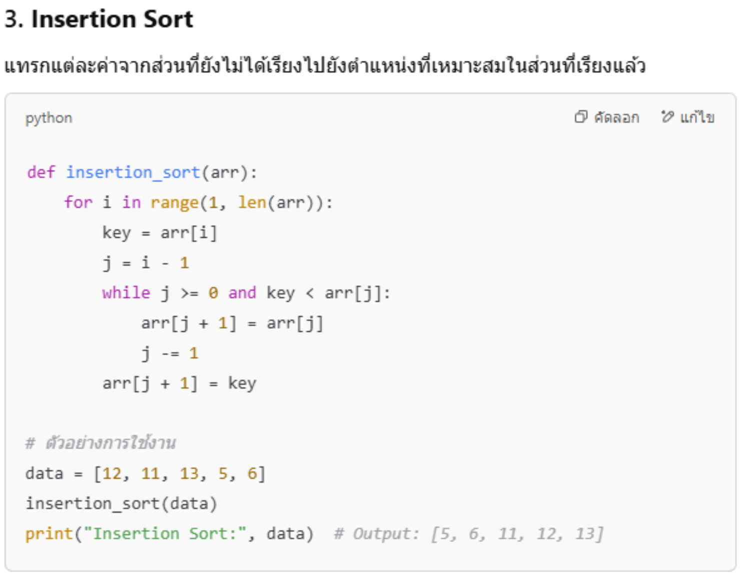
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# ตัวอย่างการใช้งาน
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("Bubble Sort:", data)  # Output: [11, 12, 22, 25, 34, 64, 90]
```



2. Selection Sort

เลือกค่าที่เล็กที่สุดจากส่วนที่ยังไม่เรียงและสลับกับตำแหน่งแรกในส่วนที่ยังไม่เรียง

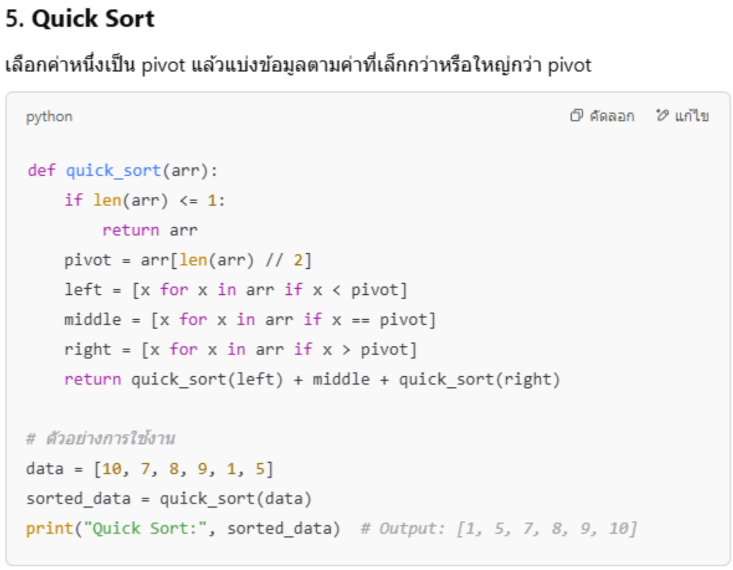
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

# ตัวอย่างการใช้งาน
data = [64, 25, 12, 22, 11]
selection_sort(data)
print("Selection Sort:", data)  # Output: [11, 12, 22, 25, 64]
```



3. Insertion Sort

แทรกแต่ละค่าจากส่วนที่ยังไม่ได้เรียงไปยังตำแหน่งที่เหมาะสมในส่วนที่เรียงแล้ว

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6]
insertion_sort(data)
print("Insertion Sort:", data)  # Output: [5, 6, 11, 12, 13]
```



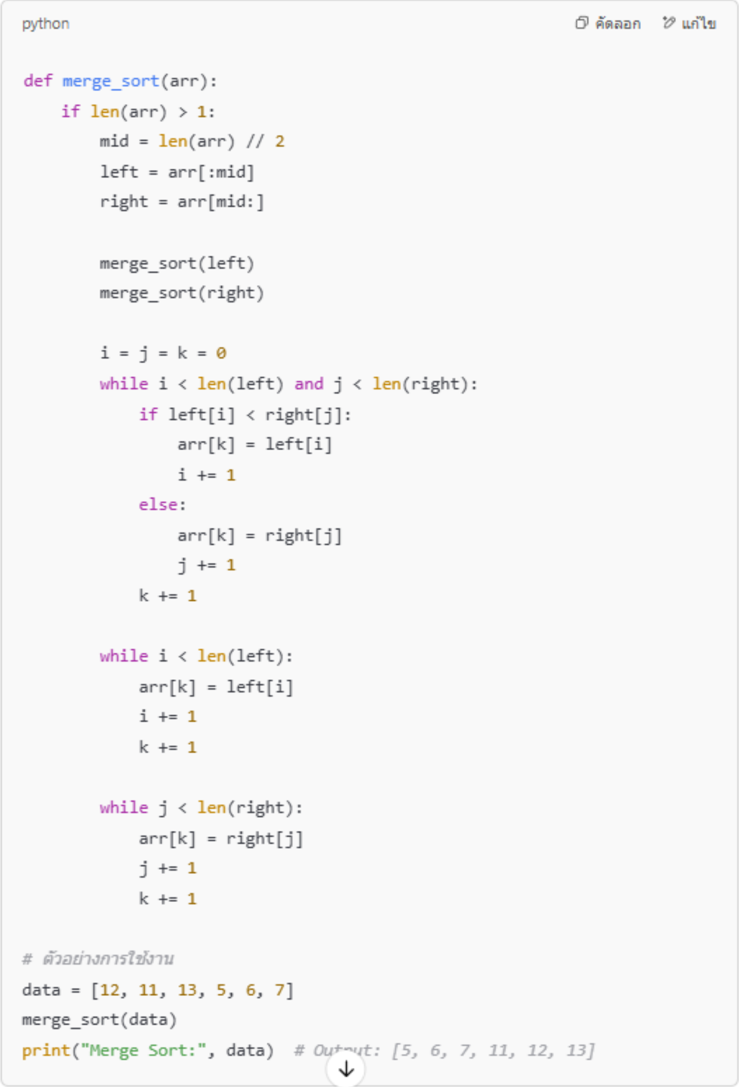
5. Quick Sort

เลือกค่าหนึ่งเป็น pivot แล้วแบ่งข้อมูลตามค่าที่เล็กกว่าหรือใหญ่กว่า pivot

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# ตัวอย่างการใช้งาน
data = [10, 7, 8, 9, 1, 5]
sorted_data = quick_sort(data)
print("Quick Sort:", sorted_data)  # Output: [1, 5, 7, 8, 9, 10]
```

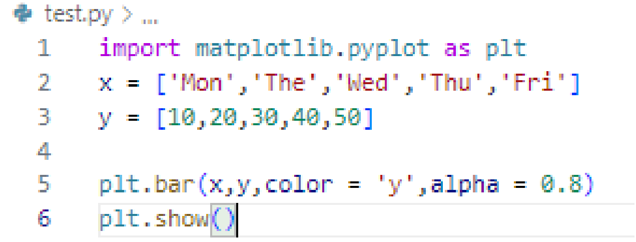
4. Merge Sort

ใช้การแบ่งและรวม (Divide and Conquer)

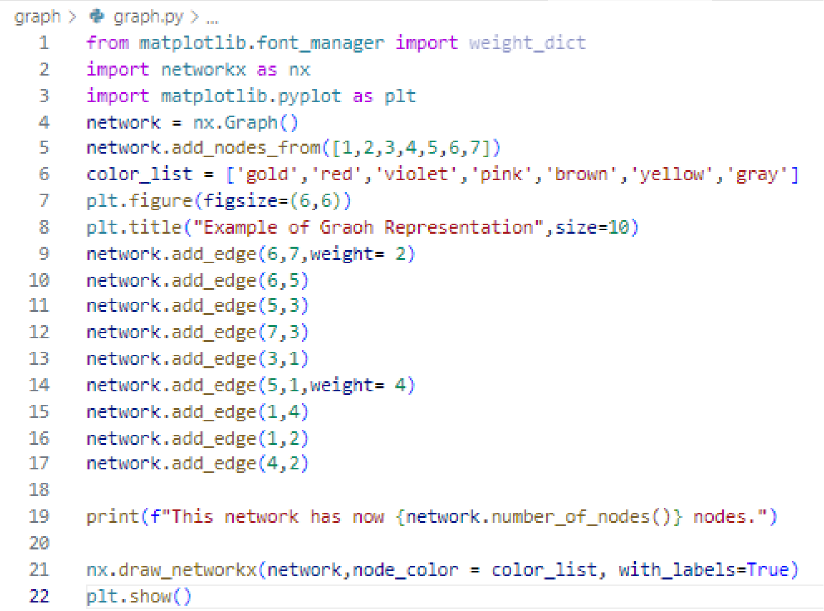


```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]

        merge_sort(left)
        merge_sort(right)

        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1

# ตัวอย่างการใช้งาน
data = [12, 11, 13, 5, 6, 7]
merge_sort(data)
print("Merge Sort:", data)  # Output: [5, 6, 7, 11, 12, 13]
```

Graph



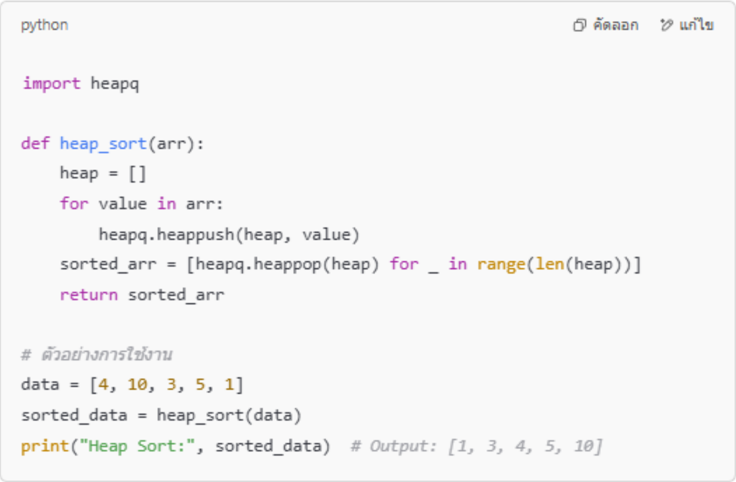
```python
import matplotlib.pyplot as plt
x = ['Mon','The','Wed','Thu','Fri']
y = [10,20,30,40,50]

plt.bar(x,y,color = 'y',alpha = 0.8)
plt.show()
```



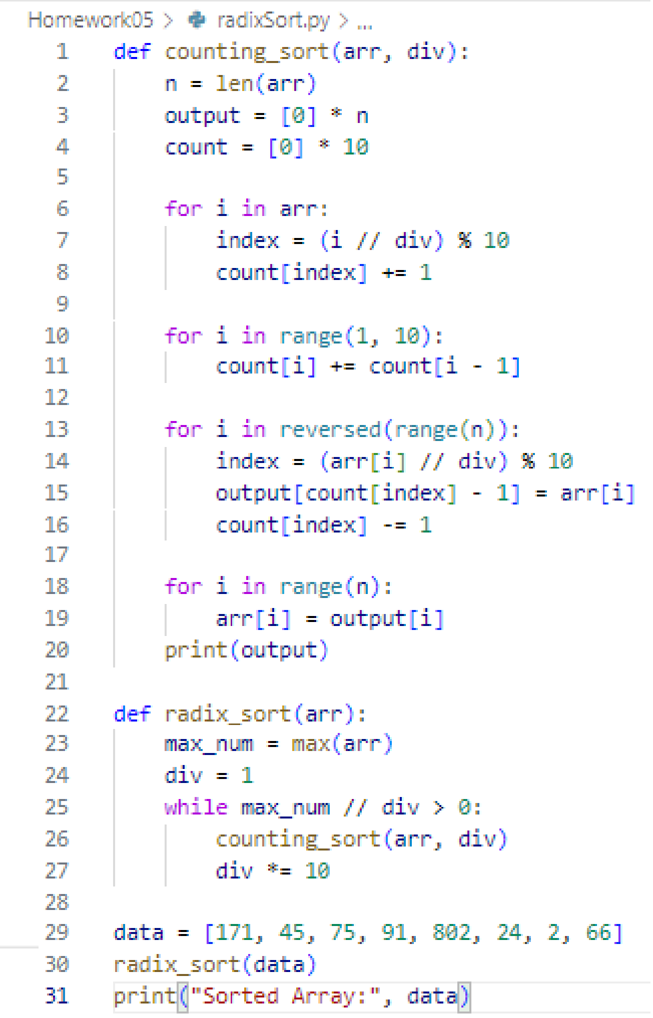
```python
from matplotlib.font_manager import weight_dict
import networkx as nx
import matplotlib.pyplot as plt
network = nx.Graph()
network.add_nodes_from([1,2,3,4,5,6,7])
color_list = ['gold','red','violet','pink','brown','yellow','gray']
plt.figure(figsize=(6,6))
plt.title("Example of Graoh Representation",size=10)
network.add_edge(6,7,weight= 2)
network.add_edge(6,5)
network.add_edge(5,3)
network.add_edge(7,3)
network.add_edge(3,1)
network.add_edge(5,1,weight= 4)
network.add_edge(1,4)
network.add_edge(1,2)
network.add_edge(4,2)

print(f"This network has now {network.number_of_nodes()} nodes.")

nx.draw_networkx(network,node_color = color_list, with_labels=True)
plt.show()
```

6. Heap Sort

ใช้โครงสร้างข้อมูล heap



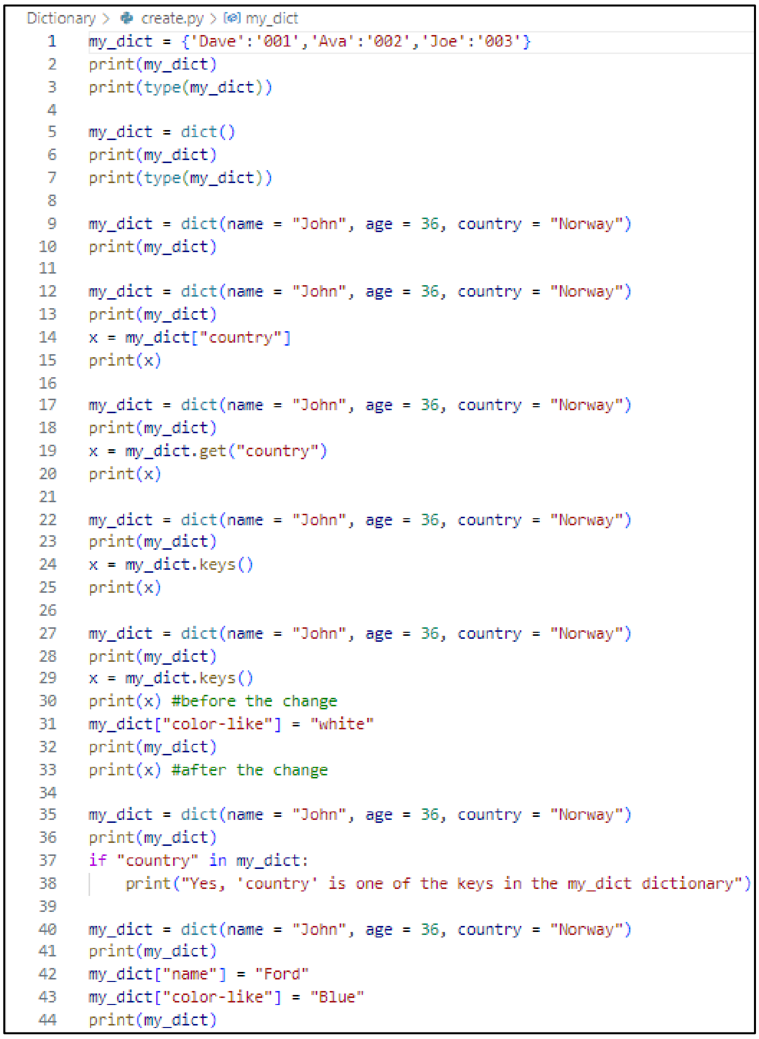
```python
import heapq

def heap_sort(arr):
    heap = []
    for value in arr:
        heapq.heappush(heap, value)
    sorted_arr = [heapq.heappop(heap) for _ in range(len(heap))]
    return sorted_arr

# ตัวอย่างการใช้งาน
data = [4, 10, 3, 5, 1]
sorted_data = heap_sort(data)
print("Heap Sort:", sorted_data)  # Output: [1, 3, 4, 5, 10]
```



```python
def counting_sort(arr, div):
    n = len(arr)
    output = [0] * n
    count = [0] * 10

    for i in arr:
        index = (i // div) % 10
        count[index] += 1

    for i in range(1, 10):
        count[i] += count[i - 1]

    for i in reversed(range(n)):
        index = (arr[i] // div) % 10
        output[count[index] - 1] = arr[i]
        count[index] -= 1

    for i in range(n):
        arr[i] = output[i]
    print(output)

def radix_sort(arr):
    max_num = max(arr)
    div = 1
    while max_num // div > 0:
        counting_sort(arr, div)
        div *= 10

data = [171, 45, 75, 91, 802, 24, 2, 66]
radix_sort(data)
print("Sorted Array:", data)
```

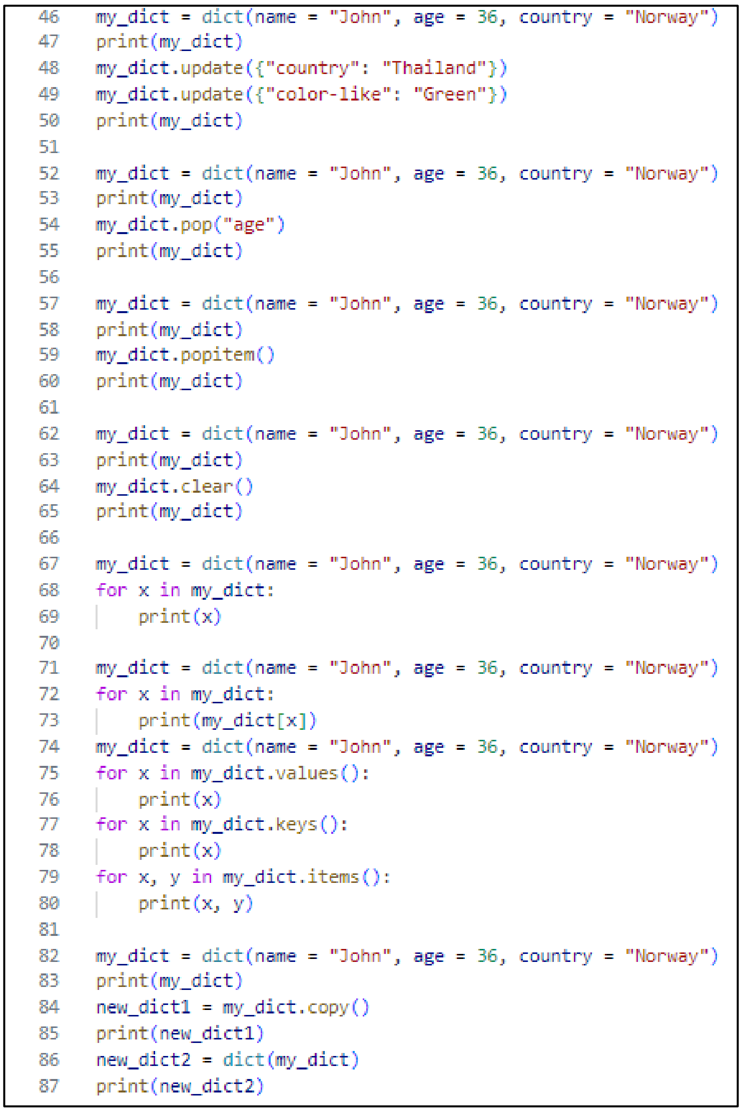
Dictionary

ตัวอย่างการใช้ Dict

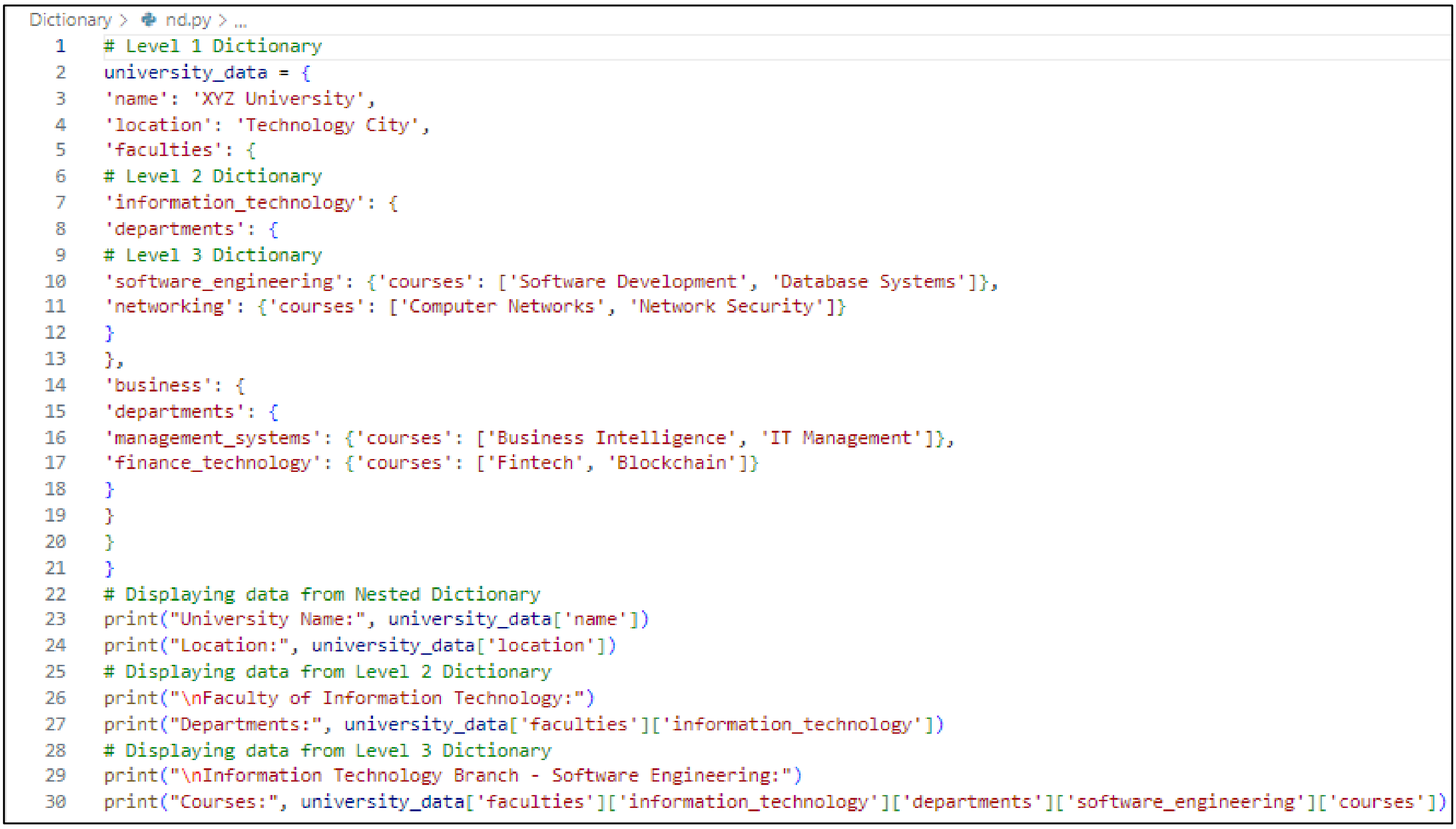


Dictionary > create.py > my_dict

```python
my_dict = {'Dave':'001','Ava':'002','Joe':'003'}
print(my_dict)
print(type(my_dict))

my_dict = dict()
print(my_dict)
print(type(my_dict))

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
x = my_dict["country"]
print(x)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
x = my_dict.get("country")
print(x)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
x = my_dict.keys()
print(x)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
x = my_dict.keys()
print(x) #before the change
my_dict["color-like"] = "white"
print(my_dict)
print(x) #after the change

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
if "country" in my_dict:
    print("Yes, 'country' is one of the keys in the my_dict dictionary")

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
my_dict["name"] = "Ford"
my_dict["color-like"] = "Blue"
print(my_dict)
```



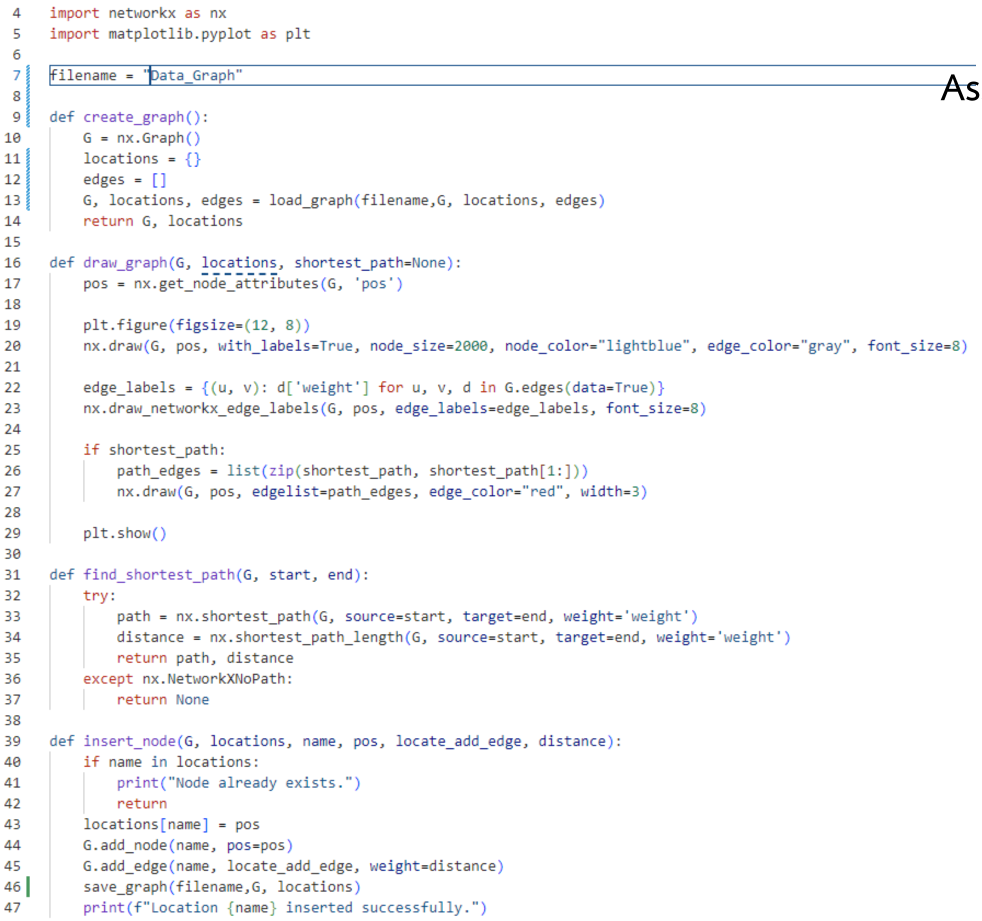
```python
my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
my_dict.update({"country": "Thailand"})
my_dict.update({"color-like": "Green"})
print(my_dict)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
my_dict.pop("age")
print(my_dict)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
my_dict.popitem()
print(my_dict)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
my_dict.clear()
print(my_dict)

my_dict = dict(name = "John", age = 36, country = "Norway")
for x in my_dict:
    print(x)

my_dict = dict(name = "John", age = 36, country = "Norway")
for x in my_dict:
    print(my_dict[x])
my_dict = dict(name = "John", age = 36, country = "Norway")
for x in my_dict.values():
    print(x)
for x in my_dict.keys():
    print(x)
for x, y in my_dict.items():
    print(x, y)

my_dict = dict(name = "John", age = 36, country = "Norway")
print(my_dict)
new_dict1 = my_dict.copy()
print(new_dict1)
new_dict2 = dict(my_dict)
print(new_dict2)
```



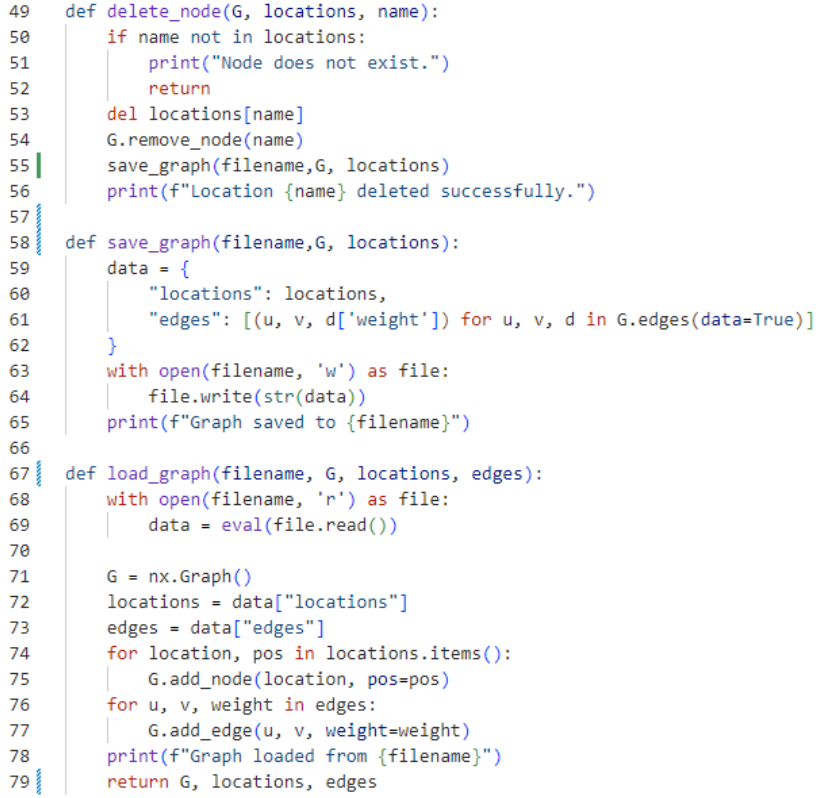
Dictionary > nd.py > ...

```python
# Level 1 Dictionary
university_data = {
'name': 'XYZ University',
'location': 'Technology City',
'faculties': {
# Level 2 Dictionary
'information_technology': {
'departments': {
# Level 3 Dictionary
'software_engineering': {'courses': ['Software Development', 'Database Systems']},
'networking': {'courses': ['Computer Networks', 'Network Security']}
}
},
'business': {
'departments': {
'management_systems': {'courses': ['Business Intelligence', 'IT Management']},
'finance_technology': {'courses': ['Fintech', 'Blockchain']}
}
}
}
}
# Displaying data from Nested Dictionary
print("University Name:", university_data['name'])
print("Location:", university_data['location'])
# Displaying data from Level 2 Dictionary
print("\nFaculty of Information Technology:")
print("Departments:", university_data['faculties']['information_technology'])
# Displaying data from Level 3 Dictionary
print("\nInformation Technology Branch - Software Engineering:")
print("Courses:", university_data['faculties']['information_technology']['departments']['software_engineering']['courses'])
```

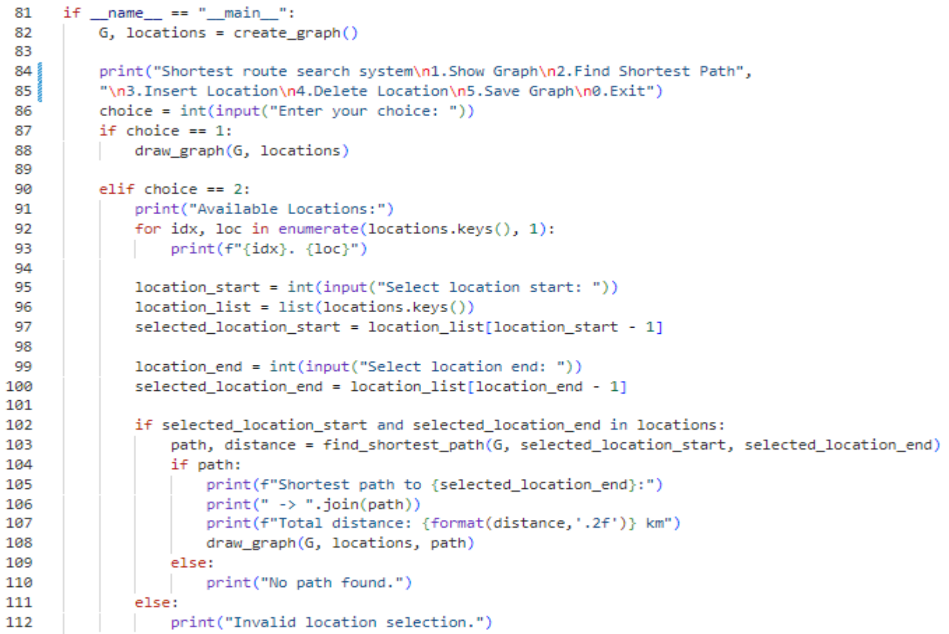
Assignmet Graph



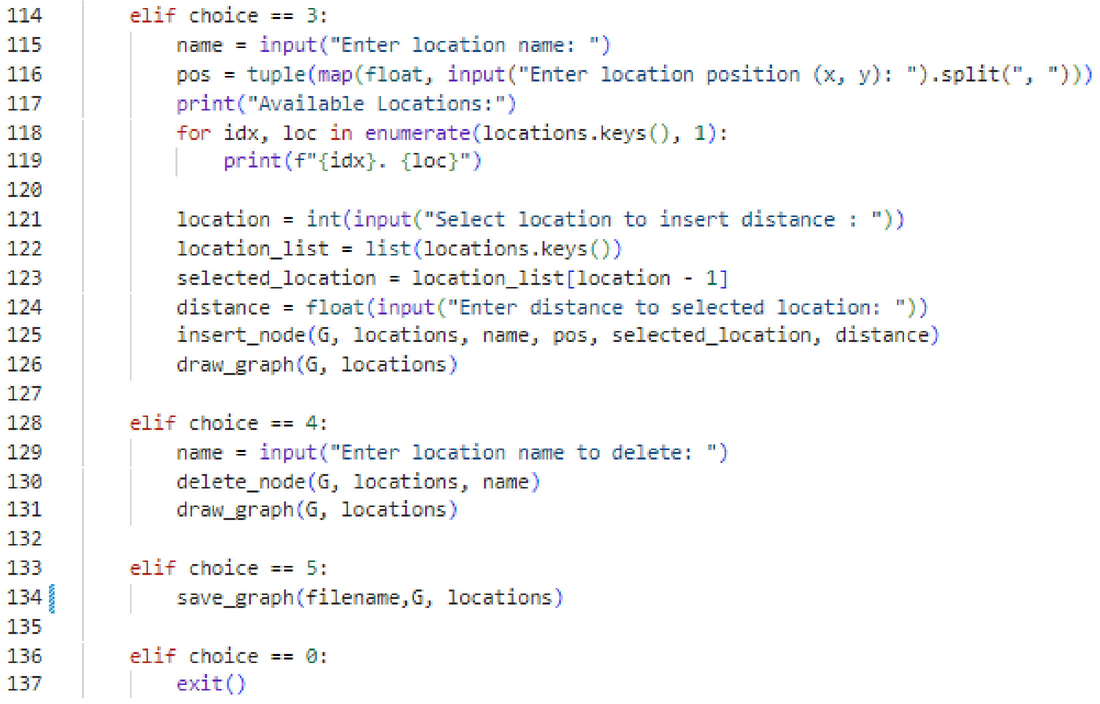
```python
import networkx as nx
import matplotlib.pyplot as plt

filename = "Data_Graph"

def create_graph():
    G = nx.Graph()
    locations = {}
    edges = []
    G, locations, edges = load_graph(filename,G, locations, edges)
    return G, locations

def draw_graph(G, locations, shortest_path=None):
    pos = nx.get_node_attributes(G, 'pos')

    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_size=2000, node_color="lightblue", edge_color="gray", font_size=8)

    edge_labels = {(u, v): d['weight'] for u, v, d in G.edges(data=True)}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)

    if shortest_path:
        path_edges = list(zip(shortest_path, shortest_path[1:]))
        nx.draw(G, pos, edgelist=path_edges, edge_color="red", width=3)

    plt.show()

def find_shortest_path(G, start, end):
    try:
        path = nx.shortest_path(G, source=start, target=end, weight='weight')
        distance = nx.shortest_path_length(G, source=start, target=end, weight='weight')
        return path, distance
    except nx.NetworkXNoPath:
        return None

def insert_node(G, locations, name, pos, locate_add_edge, distance):
    if name in locations:
        print("Node already exists.")
        return
    locations[name] = pos
    G.add_node(name, pos=pos)
    G.add_edge(name, locate_add_edge, weight=distance)
    save_graph(filename,G, locations)
    print(f"Location {name} inserted successfully.")
```



```python
def delete_node(G, locations, name):
    if name not in locations:
        print("Node does not exist.")
        return
    del locations[name]
    G.remove_node(name)
    save_graph(filename,G, locations)
    print(f"Location {name} deleted successfully.")

def save_graph(filename,G, locations):
    data = {
        "locations": locations,
        "edges": [(u, v, d['weight']) for u, v, d in G.edges(data=True)]
    }
    with open(filename, 'w') as file:
        file.write(str(data))
    print(f"Graph saved to {filename}")

def load_graph(filename, G, locations, edges):
    with open(filename, 'r') as file:
        data = eval(file.read())

    G = nx.Graph()
    locations = data["locations"]
    edges = data["edges"]
    for location, pos in locations.items():
        G.add_node(location, pos=pos)
    for u, v, weight in edges:
        G.add_edge(u, v, weight=weight)
    print(f"Graph loaded from {filename}")
    return G, locations, edges
```



```python
if __name__ == "__main__":
    G, locations = create_graph()

    print("Shortest route search system\n1.Show Graph\n2.Find Shortest Path",
    "\n3.Insert Location\n4.Delete Location\n5.Save Graph\n0.Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        draw_graph(G, locations)

    elif choice == 2:
        print("Available Locations:")
        for idx, loc in enumerate(locations.keys(), 1):
            print(f"{idx}. {loc}")

        location_start = int(input("Select location start: "))
        location_list = list(locations.keys())
        selected_location_start = location_list[location_start - 1]

        location_end = int(input("Select location end: "))
        selected_location_end = location_list[location_end - 1]

        if selected_location_start and selected_location_end in locations:
            path, distance = find_shortest_path(G, selected_location_start, selected_location_end)
            if path:
                print(f"Shortest path to {selected_location_end}:")
                print(" -> ".join(path))
                print(f"Total distance: {format(distance,'.2f')} km")
                draw_graph(G, locations, path)
            else:
                print("No path found.")
        else:
            print("Invalid location selection.")
```



```python
    elif choice == 3:
        name = input("Enter location name: ")
        pos = tuple(map(float, input("Enter location position (x, y): ").split(", ")))
        print("Available Locations:")
        for idx, loc in enumerate(locations.keys(), 1):
            print(f"{idx}. {loc}")

        location = int(input("Select location to insert distance : "))
        location_list = list(locations.keys())
        selected_location = location_list[location - 1]
        distance = float(input("Enter distance to selected location: "))
        insert_node(G, locations, name, pos, selected_location, distance)
        draw_graph(G, locations)

    elif choice == 4:
        name = input("Enter location name to delete: ")
        delete_node(G, locations, name)
        draw_graph(G, locations)

    elif choice == 5:
        save_graph(filename,G, locations)

    elif choice == 0:
        exit()
```