ASSIGNMENT-4(2)

Exercise 1: List Operations

1. Create a list called `numbers` containing the numbers `1`, `2`, `3`, `4`, and `5`.

2. Append the number `6` to the list.

3. Remove the number `3` from the list.

4. Insert the number `0` at the beginning

Program:

```
numbers = [1, 2, 3, 4, 5]

numbers.append(6)

numbers.remove(3)

numbers.insert(0, 0)

print(numbers)
```

Exercise 2: Tuple Operations

1. Create a tuple called `coordinates` containing the elements `10.0`, `20.0`, and `30.0`.

2. Access and print the second element of the tuple.

3. Try to change the third element of the tuple to `40.0`. What happens?

Program:

```
coordinates = (10.0, 20.0, 30.0)

print(coordinates[1])
```

** Attempting to change the third element would result in an error since tuples are immutable.

Exercise 3: Set Operations

1. Create a set called `fruits` containing `"apple"`, `"banana"`, `"cherry"`.

2. Add `"orange"` to the set.

3. Remove `"banana"` from the set.

4. Check if `"cherry"` is in the set and print a message based on the result.

5. Create another set called `citrus` with elements `"orange"`, `"lemon"`, `"lime"`.

6. Perform a union of `fruits` and `citrus` and print the result.

7. Perform an intersection of `fruits` and `citrus` and print the result.

Program:

```
fruits = {"apple", "banana", "cherry"}

fruits.add("orange")

fruits.remove("banana")

print("cherry" in fruits)


citrus = {"orange", "lemon", "lime"}

print(fruits.union(citrus))

print(fruits.intersection(citrus))
```

Exercise 4: Dictionary Operations

1. Create a dictionary called `person` with keys `"name"`, `"age"`, and `"city"`, and values `"John"`, `30`, and `"New York"`, respectively.

2. Access and print the `"name"` key from the dictionary.

3. Update the `"age"` key to `31`.

4. Add a new key-value pair `"email": "john@example.com"` to the dictionary.

5. Remove the `"city"` key from the dictionary.

6. Print the final dictionary.

Program:

```
person = {"name": "John", "age": 30, "city": "New York"}

print(person["name"])

person["age"] = 31

person["email"] = "john@example.com"

del person["city"]

print(person)
```

Exercise 5: Nested Dictionary

1. Create a dictionary called `school` where the keys are student names and the values are dictionaries containing the subjects and their corresponding grades. Example structure:

  ```python
  school = {
     "Alice": {"Math": 90, "Science": 85},
     "Bob": {"Math": 78, "Science": 92},
     "Charlie": {"Math": 95, "Science": 88}
  }
  ```

2. Print the grade of `"Alice"` in `"Math"`.

3. Add a new student `"David"` with grades `"Math": 80` and `"Science": 89`.

4. Update `"Bob"`'s `"Science"` grade to 95.

5. Print the final `school` dictionary.

Program:

```
school = {
   "Alice": {"Math": 90, "Science": 85},
   "Bob": {"Math": 78, "Science": 92},
   "Charlie": {"Math": 95, "Science": 88}
}
print(school["Alice"]["Math"])
school["David"] = {"Math": 80, "Science": 89}
school["Bob"]["Science"] = 95
print(school)
```

Exercise 6: List Comprehension

1. Given a list of numbers `[1, 2, 3, 4, 5]`, use list comprehension to create a new list where each number is squared.

2. Print the new list.

Program:

```
numbers = [1, 2, 3, 4, 5]

squared_numbers = [x**2 for x in numbers]

print(squared_numbers)
```

Exercise 7: Set Comprehension

1. Create a set comprehension that generates a set of squared numbers from the list `[1, 2, 3, 4, 5]`.

2. Print the resulting set.

Program:

```
numbers = [1, 2, 3, 4, 5]

squared_set = {x**2 for x in numbers}

print(squared_set)
```

Exercise 8: Dictionary Comprehension

1. Create a dictionary comprehension that generates a dictionary where the keys are the numbers from `1` to `5`, and the values are the cubes of the keys.

2. Print the resulting dictionary.

Program:

```
cubes = {x: x**3 for x in range(1, 6)}

print(cubes)
```

Exercise 9: Combining Collections

1. Create two lists: `keys = ["name", "age", "city"]` and `values = ["Alice", 25, "Paris"]`.

2. Use the `zip()` function to combine the `keys` and `values` lists into a dictionary.

3. Print the resulting dictionary.

Program:

```
keys = ["name", "age", "city"]

values = ["Alice", 25, "Paris"]
```

result = dict(zip(keys, values))

print(result)

Exercise 10: Count Word Occurrences (Using a Dictionary)

1. Write a Python program that takes a string as input and counts the occurrences of each word in the string using a dictionary. Example input:

   ```python
   sentence = "the quick brown fox jumps over the lazy dog the fox"
   ```

2. Print the resulting dictionary with word counts.

Program:

```
sentence = "the quick brown fox jumps over the lazy dog the fox"
word_list = sentence.split()
word_count = {}
for word in word_list:
    if word in word_count:
        word_count[word] += 1
    else:
        word_count[word] = 1
print(word_count)
```

Exercise 11: Unique Elements in Two Sets

1. Create two sets: `set1 = {1, 2, 3, 4, 5}` and `set2 = {4, 5, 6, 7, 8}`.

2. Find and print the unique elements in both sets combined.

3. Find and print the common elements between the two sets.

4. Find and print the elements that are only in `set1` but not in `set2`.

Program:

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
```

print(set1.union(set2))

print(set1.intersection(set2))

print(set1.difference(set2))

Exercise 12: Tuple Unpacking

1. Create a tuple with three elements: `("Alice", 25, "Paris")`.

2. Unpack the tuple into three variables: `name`, `age`, and `city`.

3. Print the variables to verify the unpacking.

Program:

data = ("Alice", 25, "Paris")

name, age, city = data

print(name, age, city)

Exercise 13: Frequency Counter with Dictionary

1. Write a Python program that counts the frequency of each letter in a given string using a dictionary. Example string:

   ```python
   text = "hello world"
   ```

2. Print the resulting dictionary with letter frequencies.

Program:

text = "hello world"

frequency = { }

for letter in text:

    frequency[letter] = frequency.get(letter, 0) + 1

print(frequency)

Exercise 14: Sorting a List of Tuples

1. Given a list of tuples representing students and their grades:

```python
students = [("Alice", 90), ("Bob", 80), ("Charlie", 85)]
```

2. Sort the list by grades in descending order and print the sorted list.

Program:

```
students = [("Alice", 90), ("Bob", 80), ("Charlie", 85)]
for i in range(len(students)):
    for j in range(i + 1, len(students)):
        if students[i][1] < students[j][1]:
            students[i], students[j] = students[j], students[i]
print(students)
```