Python Data Structures: Practice Worksheet

1. Tuples and Sequences

Tuples are ordered, immutable collections that can store elements of different data types.

Key Operations:

- Creation: Using parentheses `()`
- Indexing: Accessing elements using zero-based indices
- Unpacking: Assigning tuple elements to variables
- Methods: `count()`, `index()`

Examples:

python

```python
# Creating tuples
colors = ("red", "green", "blue")
mixed_tuple = (1, "hello", 3.14, True)

# Indexing (positive and negative)
first_color = colors[0]  # "red"
last_color = colors[-1]  # "blue"

# Tuple unpacking
r, g, b = colors

# Empty and singleton tuples
empty = ()
singleton = (42,)  # Note the comma is needed for single-element tuples

# Tuple methods
colors = ("red", "green", "blue", "red", "yellow")
red_count = colors.count("red")  # 2
green_index = colors.index("green")  # 1

# Nested tuples
nested = ((1, 2), (3, 4))
```

Practice Assignment 1:

Write a function that takes a list of tuples representing (x, y) coordinates and returns the tuple with the largest distance from the origin (0, 0).

2. Sets

Sets are unordered collections of unique elements.

Key Operations:

- Creation: Using curly braces `{}` or `set()`
- Membership: `in` operator
- Methods: `add()`, `remove()`, `discard()`, `update()`
- Set operations: `union()`, `intersection()`, `difference()`, `symmetric_difference()`

Examples:

python

```python
# Creating sets
fruits = {"apple", "banana", "cherry"}
even_numbers = {2, 4, 6, 8, 10}

# Note: {} creates an empty dictionary, not an empty set
empty_set = set()

# Adding and removing elements
fruits.add("orange")
fruits.remove("banana")  # Raises KeyError if element doesn't exist
fruits.discard("kiwi")   # No error if element doesn't exist

# Set operations
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}

union = set_a | set_b  # or set_a.union(set_b)
intersection = set_a & set_b  # or set_a.intersection(set_b)
difference = set_a - set_b  # or set_a.difference(set_b)
symmetric_difference = set_a ^ set_b  # or set_a.symmetric_difference(set_b)

# Set comprehensions
squares = {x**2 for x in range(10)}
```

Practice Assignment 2:

Write a function that takes two lists and returns a tuple containing:

1. A set of elements common to both lists
2. A set of elements unique to the first list
3. A set of elements unique to the second list

3. Dictionaries

Dictionaries are mutable, unordered collections of key-value pairs.

Key Operations:

- Creation: Using curly braces with colons `{key: value}`
- Accessing: Using keys as indices
- Methods: `keys()`, `values()`, `items()`, `get()`, `update()`

Examples:

python

```python
# Creating dictionaries
person = {"name": "Alice", "age": 30, "city": "New York"}
empty_dict = {}

# Accessing and modifying values
name = person["name"]  # "Alice"
person["age"] = 31

# Using get() to provide default value
country = person.get("country", "Unknown")  # "Unknown"

# Adding and removing items
person["job"] = "Engineer"
del person["city"]

# Dictionary methods
keys = person.keys()
values = person.values()
items = person.items()  # Returns tuples of (key, value) pairs

# Updating dictionaries
person.update({"age": 32, "married": True})

# Dictionary comprehensions
squares = {x: x**2 for x in range(5)}
```

Practice Assignment 3:

Write a function that takes a string as input and returns a dictionary where keys are characters and values are the number of times each character appears in the string (character frequency).

4. Sequence Operations

Common operations across sequence types (lists, tuples, strings).

Key Operations:

- Indexing and slicing
- Concatenation and repetition
- Membership testing
- Iteration

- Length, minimum, maximum

Examples:

```python
# Operations common to all sequences
sequence = [1, 2, 3, 4, 5]

# Indexing and slicing
first = sequence[0]
last = sequence[-1]
subset = sequence[1:3]  # [2, 3]

# Concatenation and repetition
combined = [1, 2, 3] + [4, 5, 6]  # [1, 2, 3, 4, 5, 6]
repeated = [1, 2, 3] * 3  # [1, 2, 3, 1, 2, 3, 1, 2, 3]

# Membership testing
is_present = 3 in sequence  # True

# Built-in functions
length = len(sequence)
minimum = min(sequence)
maximum = max(sequence)

# Enumerate for getting index and value
for i, value in enumerate(sequence):
    print(f"Index {i}: {value}")
```

Practice Assignment 4:

Write a function that takes a sequence (list, tuple, or string) and returns a dictionary that maps each unique element to all the indices where it appears in the sequence.

Challenging Problems

Problem 1: Set and Dictionary Operations

Write a function that takes a list of dictionaries (each representing a person with 'name' and 'skills' keys, where 'skills' is a list of strings) and returns:

1. A set of all unique skills across all people
2. A dictionary mapping each skill to a list of people who have that skill

Problem 2: Advanced Tuple Packing/Unpacking

Write a function that performs a "matrix transposition" by taking a tuple of tuples (representing a matrix) and returning a new tuple of tuples where rows and columns are swapped.

Problem 3: Dictionary Nesting

Create a function that processes student data in the form of a list of dictionaries. Each dictionary contains a student's name, grade, and scores in different subjects. The function should return a nested dictionary where:

- The outer keys are grades
- The inner keys are student names
- The values are the average scores for each student

Problem 4: Sequence Alignment

Write a function that takes two sequences (strings, lists, or tuples) and finds the longest common subsequence between them.

Problem 5: Dictionary and Set Combination

Create a function that analyzes a dictionary of products where keys are product IDs and values are dictionaries containing:

- name: product name
- category: product category
- price: product price

The function should return a dictionary where keys are categories and values are sets of product IDs in that category, sorted by price (highest to lowest).

Best Practices and Common Pitfalls

Tuples:

✅ Use tuples for immutable data or when returning multiple values

✅ Use tuple unpacking to assign multiple variables at once

❌ Don't try to modify tuple elements

❌ Don't rely on tuple mutability; they're immutable on the outside but can contain mutable objects

Sets:

✅ Use sets for membership testing and removing duplicates

✅ Use set operations for mathematical operations between collections

❌ Don't assume sets maintain order (they're unordered)

❌ Don't use `{}` to create empty sets (that creates an empty dictionary)

Dictionaries:

✅ Use dictionaries when you need key-value mappings

✅ Use `get()` method to provide default values

✅ Use dictionary comprehensions for concise creation

❌ Don't forget that dictionary keys must be immutable (no lists as keys)

❌ Don't assume dictionaries maintain insertion order (they do in Python 3.7+, but it's good practice not to rely on this)

Sequences:

✅ Use slicing notation for extracting parts of sequences

✅ Use the common sequence operations consistently across different types

❌ Don't modify a sequence while iterating over it

❌ Don't use index-based iteration when direct iteration is cleaner