coding temple

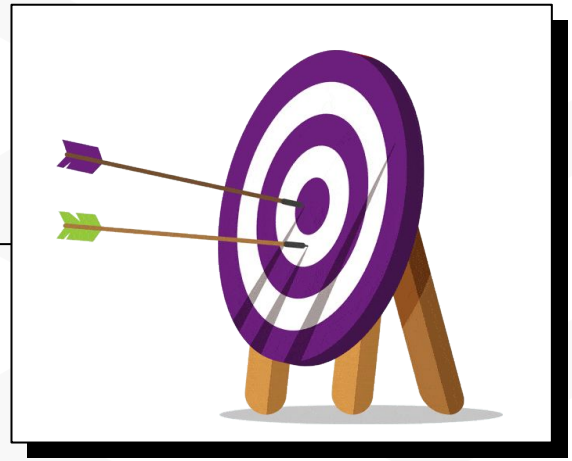
Software Engineering

Module 3: Python Intermediate

Python Sets

Learning Objectives Students should be able to:

- Define and understand the unique characteristics of Python sets, including their unordered, unchangeable, and unindexed nature, and the prevention of duplicate elements.
- Create and initialize sets in Python, using both literal notation and the `set()` constructor, and convert other data types into sets.
- Access and iterate over elements in a set, understanding the constraints due to the unchangeable and unindexed nature of sets.
- Apply methods to add elements to sets (`add`, `update`) and understand the constraints on adding elements, such as the immutability of set elements.
- Employ various methods to remove elements from sets (`remove`, `discard`, `pop`, `clear`), and articulate the differences between these methods.
- Perform standard set operations such as union, intersection, difference, and symmetric difference, and provide practical examples of these operations.
- Use set comprehensions in Python for efficient and expressive set creation, applying conditions and transformations within the comprehension.
- Identify and avoid common mistakes when working with sets and apply best practices for efficient and effective set usage.
- Explain advanced set methods such as `issubset`, `issuperset`, and `isdisjoint`, and apply these methods in practical contexts.
- Discuss various real-world applications and use cases of Python sets, demonstrating an understanding of how sets can be utilized in different programming scenarios.



Exploring Python Sets: A Party Where Every Element is Unique!

The Unordered Social Mixer: elements don't have a fixed position
Unique Guests Only, Please!: Unique characteristic - no duplicates rule
The Unchangeable RSVP List: Once an element is in a set, it is unchangeable
No Numbered Invitations: Set elements are unindexed

SETS vs Lists Tuples Dictionaries

CHARACTERISTIC	SETS	LISTS or TUPLES	DICTIONARIES
Uniqueness	**Eliminate duplicates**	Does not eliminate	
Performance	**Faster**	No so fast!	
Simplicity in Uniqueness operations	**Operations like union, intersection and difference is straightforward**	No straightforward operations	
No key-Value Pairs Needed	**No value Associating additional information**		**Associating additional information**
Set-Specific Operations	**Perform set operations**		Operations that aren´t directly available or efficient

Creating and Initializing Sets in Python: Joining the Unique Party!

Hey, ready to create your own exclusive, unique-element party, also known as a Python set? In this chapter, we'll learn how to create and initialize sets, and how to invite guests (elements) from other data types to our set party. It's like sending out those special invitations and ensuring everyone who shows up is distinct.

Let's start our set creation adventure!

Creating Your First Set: Creating a set in Python is like planning a new party. It's simple and straightforward.

Using Curly Braces {}: This is like creating a guest list with the names Alice, Bob, and Charlie. Remember, no duplicates are allowed!

```
unique_party = {'Alice', 'Bob', 'Charlie'}
```

Using the set() Constructor: This is like having a list of potential guests and then deciding, "Let's make this a unique-name party!"

```
guest_list = set(['Alice', 'Bob', 'Charlie'])
```

Converting Other Data Types to Sets: How to turn a list, tuple or dictionary into a set party.

From a List: This is like taking a regular party list and converting it into our unique-name party. The duplicate 'Alice' gets automatically removed.

```
list_guests = ['Alice', 'Bob', 'Alice', 'Diana']
set_party = set(list_guests)
```

From a Tuple: Like waving a magic wand and transforming a formal event (tuple) into our cool set party.

```
tuple_guests = ('Alice', 'Bob', 'Charlie')
set_party = set(tuple_guests)
```

Converting Other Data Types to Sets: How to turn a list, tuple or dictionary into a set party. (Cont.)

From a Dictionary: Here, we're inviting only the values from a dictionary, ensuring they're unique at our set party.

```
dict_guests = {'name1': 'Alice', 'name2': 'Bob', 'name3': 'Alice'}
set_party = set(dict_guests.values())
```

Remember, in Python, a set cannot contain another set directly as its element. This is because sets in Python are designed to only contain hashable objects, and sets themselves are not hashable. The unhashability of sets is due to their mutable nature; their contents can change, which makes it impossible for them to have a consistent hash value.

Peeking into the Party: Accessing Elements in Python Sets

Imagine finding someone at our unique-name party without a guest list. That's right, in the world of sets, there's no numbered invitation or seating chart. This is because sets are unchangeable and unindexed. Once a guest (element) is at the party, you can't change their name (alter the element). And without a guest list (index), you can't just jump to the fifth person in line.
The following methods let you explore who's in attendance.

Peeking into the Party: Accessing Elements in Python Sets

Using Loops to Mingle with Guests: Just as you stroll through the party to see who's there, you can loop through a set to access its elements.

```
for guest in {'Alice', 'Bob', 'Charlie'}:
      print("Welcome,", guest, "to the party!")
```

Peeking into the Party: Accessing Elements in Python Sets

The 'in' Keyword: Is Your Friend at the Party?**:** If you're wondering whether a specific guest is at the party, use the in keyword.

```
guests = {'Alice', 'Bob', 'Charlie'}
if 'Alice' in guests:
        print("Alice is enjoying the party!")
else:
        print("Alice couldn't make it.")
```

Welcoming New Guests: Adding and Updating Elements in Python Sets

Just like deciding to invite more friends to our exclusive party, in Python sets, you can always add new and unique guests. But remember, no duplicates are allowed at this party! Let's explore how to expand our guest list with `add` and `update` methods:

The Rule of Uniqueness: Constraints on Adding to Sets

- **Immutability of Elements:** Elements can be removed or added. They cannot change or be modified.
- **No Duplicates Allowed:** Duplicated elements have NO effect

Welcoming New Guests: Adding and Updating Elements in Python Sets

add Method: Suppose you want to invite 'Diana' to the party. Here's how you do it.

```
guests = {'Alice', 'Bob', 'Charlie'}
guests.add('Diana')
print(guests)
```

Let's watch a video for better understanding.

Welcoming New Guests: Adding and Updating Elements in Python Sets (Cont.)

`update` Method: What if you want to invite a group of friends at once? Use `update`!

```
guests = {'Alice', 'Bob', 'Charlie'}
more_guests = ['Ethan', 'Fiona']
guests.update(more_guests)
print(guests)
```

Let's watch a video for better understanding.

Saying Goodbye: Removing Elements from Python Sets

Even the best parties sometimes need to wind down, and the same goes for our Python set party. Whether it's time for certain guests to leave or to clear out the party entirely, Python sets provide various ways to manage this.

Choosing the Right Method: Best Practices in Set Management

- **REMOVE**: Elements are in the set for sure.
- **DISCARD**: Not sure if the element is in the set or not
- **POP**: Need to remove an element, no matter which one.
- **CLEAR**: Delete every element and start fresh

Saying Goodbye: Removing Elements from Python Sets

`remove` **Method -** Specific Goodbyes: Imagine you need to politely ask 'Charlie' to leave the party:

```
guests = {'Alice', 'Bob', 'Charlie'}
guests.remove('Diana')
print(guests)
```

Saying Goodbye: Removing Elements from Python Sets

`discard` **Method -** The Polite Removal: Now, what if you're not sure if 'Diana' is at the party? Use discard

```
guests.discard('Diana')
```

Let's watch the next video.

Saying Goodbye: Removing Elements from Python Sets

pop Method - A Surprise Goodbye: Want to randomly ask a guest to leave? pop is your method.

```
random_guest = guests.pop()
print("We randomly asked", random_guest, "to leave.")
```

Pay attention to the following video.

Saying Goodbye: Removing Elements from Python Sets

`clear` Method - Clearing the Party with `clear`

And finally, when the party's over, and everyone needs to leave, it is better to use the `clear` method

```
guests.clear()
print(guests)
```

Please click in the next video.

Mastering the Art of Python Set Operations

In the vibrant world of Python sets, we have four fundamental operations that are similar to the social dynamics at different parties. Let's delve deeper into each of these operations, understanding their nuances and practicalities.

Union: The Grand Gathering

Name	Union
What?	Joining 2 parties' guest list, ensuring no one is invited twice.
Practical Use	Merging two datasets while ensuring no duplicate entries

How it works	`set1 = {'Alice', 'Bob', 'Charlie'}`
	`set2 = {'David', 'Emma', 'Charlie'}`
	`grand_gathering = set1.union(set2)`
	`print(grand_gathering) # Output: {'Alice', 'Bob', 'Charlie', 'David', 'Emma'}`

Intersection: Finding Common Ground

Name	`Intersection`
What?	Identifying the same guests in 2 different party lists.
Practical Use	finding common elements in datasets

How it works	`set1 = {'Alice', 'Bob', 'Charlie'}`
	`set2 = {'David', 'Emma', 'Charlie'}`
	`mutual_friends = set1.intersection(set2)`
	`print(mutual_friends) # Output: {'Charlie'}`

Difference: The Exclusive Aspect

Name	`Difference`
What?	Identifying the exclusive attendees
Practical Use	Identifying unique features

How it works	`set1 = {'Alice', 'Bob', 'Charlie'}`
	`set2 = {'David', 'Emma', 'Charlie'}`
	`exclusive_guests = set1.difference(set2)`
	`print(exclusive_guests) # Output: {'Alice', 'Bob'}`

Symmetric Difference: Identifying Uniqueness

Name	Symmetric Difference
What?	*Finds people who are in one of the 2 parties going on*
Practical Use	*Compares datasets to find elements exclusive to each*

How it works	`set1 = {'Alice', 'Bob', 'Charlie'}`
	`set2 = {'David', 'Emma', 'Charlie'}`
	`unique_attendees = set1.symmetric_difference(set2)`
	`print(unique_attendees) # Output: {'Alice', 'Bob', 'David', 'Emma'}`

Practical Tips and Tricks for Set Operations

- **Union for Merging:** union. Collection from multiple sets without worrying about duplicates.
- **Intersection for Commonalities:** intersection. Pinpoint similarities between sets.
- **Difference for Uniqueness:** difference. Highlight the unique aspects of a set.
- **Symmetric Difference for Exclusive Analysis:** symmetric difference. Extract elements that are exclusive to each set.

Dancing Through the Set Party: Looping Through Python Sets

In our previous exploration of "Peeking into the Party," we delved into the basics of looping through sets in Python, akin to mingling at a social gathering and acknowledging each guest. This method is invaluable when you want to process or interact with every element in a set. Now, let's take this a step further and incorporate enumerate into our looping.

Dancing Through the Set Party: Looping Through Python Sets

Using enumerate: Keeping Track of Your Introductions

- **The Technique**: Use enumerate to loop through the set with an index.

```
guests = {'Alice', 'Bob', 'Charlie'}
for index, guest in enumerate(guests):
print("Guest #", index, "is", guest)
```

Dancing Through the Set Party: Looping Through Python Sets

Bringing order to chaos

- **Keeping Tabs:** enumerate to keep track of iterations for logging, debugging, or performing operations of amounted elements processed.
- **Versatile Applications:** Generating a list of items with their indices for display purposes or mapping set elements to another list.

```
guests = {'Alice', 'Bob', 'Charlie'}
for index, guest in enumerate(sorted(guests)):
print("Welcome, guest #", index, ": ", guest)
```

Crafting the Guest List with Style: Set Comprehensions in Python

Just as a skilled host can effortlessly create a guest list that's both exclusive and exciting, set comprehensions in Python allow you to build sets in a concise and expressive manner. It's like drafting the perfect invitation list with a few strokes of your pen. Let's dive into the elegant world of set comprehensions and see how they can make your Python programming more stylish and efficient.

Set Comprehensions: The Efficient Way to Build Sets

Set Comprehensions

- **The Concept:** make sets quickly using existing lists or sequences, and you can add conditions or changes if you want.
- **Why use them?:** Provide a clearer and more efficient method for forming sets, especially when incorporating conditions or operations on the elements.
- **Unique by Nature:** automatic elimination of duplicate elements.

```
square_party = {a**2 for a in range(6)}
print(square_party) # Output: {0, 1, 4, 9, 16, 25}
```

Set Comprehensions: Adding a Twist with Conditions

- **The Sophistication:** You can add conditions to your set comprehensions to filter elements

```
odd_square_party = {a**2 for a in range(6) if a % 2 != 0}
print(odd_square_party) # Output: {1, 9, 25}
```

Set Comprehensions: Adding a Twist with Conditions - Layering Criteria

Layering Criteria: You can layer multiple conditions to refine your set.

```
special_set = {a for a in range(16, 35) if a % 2 == 0 if a % 3 == 0}
print(special_set) # Output: {24, 18, 30}
```

Set Comprehensions: Adding a Twist with Conditions - Altering Elements

Altering Elements: You can transform elements in a specific way, based on certain criteria.

```
transformed_set = {a**2 if a % 2 == 0 else a for a in range(10)}
print(transformed_set) # Output: {0, 1, 4, 3, 16, 5, 36, 7, 64, 9}
```

Best Practices for Set Comprehensions

- **Clarity is Key:** Prioritize readability.
- **Avoid Overcomplicating:** Try not to nest multiple comprehensions or use complex conditions.
- **Remember the Uniqueness:** Set comprehensions are ideal where unique elements are a priority.

Navigating the Set Party Without a Hitch: Avoiding Pitfalls and Embracing Best Practices

Just like there are social blunders to avoid at a party, there are common pitfalls you might encounter when working with Python sets. Being aware of these can save you from coding headaches and ensure your set operations run smoothly.

Navigating the Set Party Without a Hitch: Avoiding Pitfalls and Embracing Best Practices

coding temple

Mistaking Sets for Lists or Tuples

The Misstep: We can confuse sets with lists, and tuples as they might look similar. But remember, sets are unordered and don't allow duplicates.

```
# Mistakenly using a list instead of a set
party_items = ['cake', 'balloons', 'cake'] # A list, not a set
unique_items = set(party_items)
print(unique_items) # Output: {'cake', 'balloons'}
```

Navigating the Set Party Without a Hitch: Avoiding Pitfalls and Embracing Best Practices

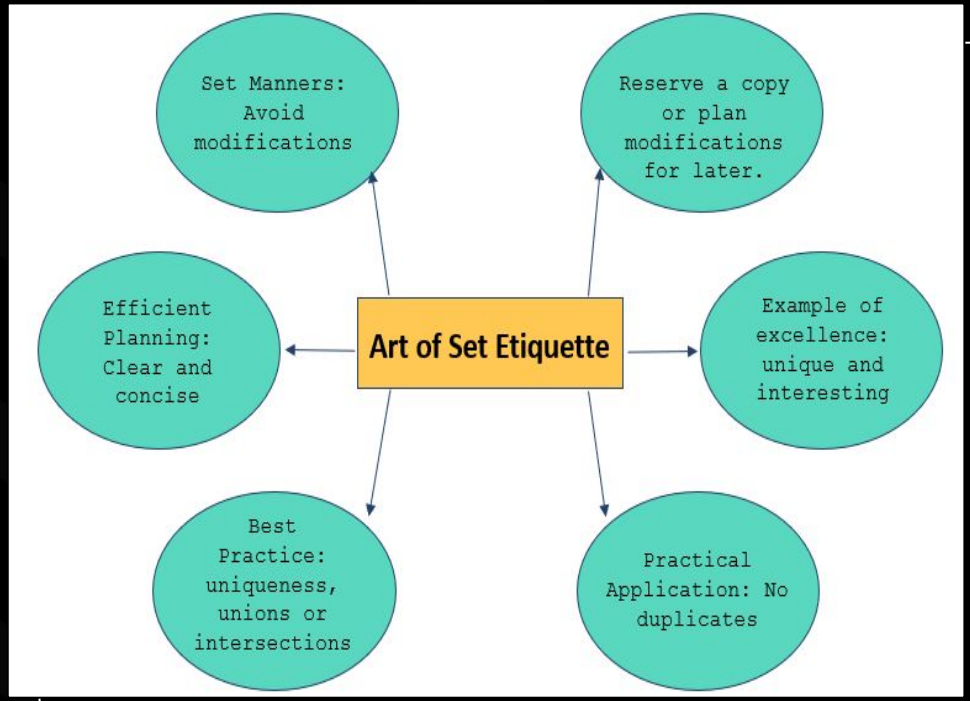
coding temple

Modifying Sets During Iteration

The Faux Pas: Trying to add or remove elements from a set while iterating over it can lead to unexpected behavior or errors.

```
guests = {'Alice', 'Bob', 'Charlie'}
for guest in guests:
      if guest == 'Alice':
            guests.remove(guest) # Risky operation!
```

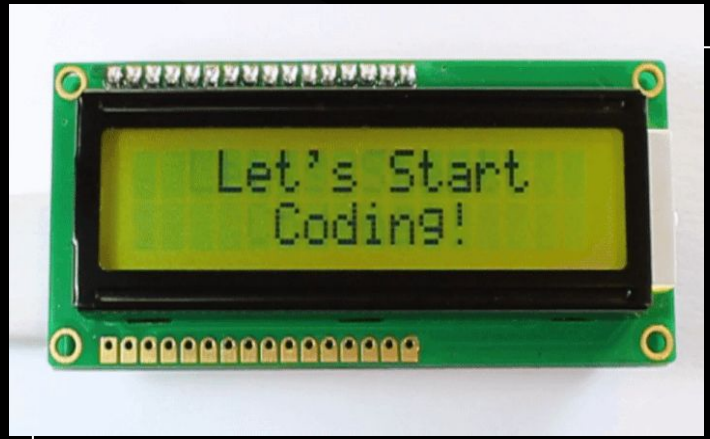
The Art of Set Etiquette: Best Practices for Set Usage



To ensure your set operations are both efficient and effective, here are some best practices to embrace:

The Art of Set Etiquette: Best Practices for Set Usage

💡 Hey there! Before we unveil our solutions, how about taking a shot at it yourself? Keep in mind that we've provided some broad strokes, so a detailed approach will help fine-tune the solution. But, this is all in good fun and a chance for you to flex those problem-solving muscles. Give it a go, and remember, it's all about learning and improving!



Exercise 1: E-commerce Product Categorization Tool

Objective:
Develop a Python program that helps an e-commerce platform categorize its products efficiently. The program should take a list of product names and categories, remove any duplicates, and organize them into distinct categories for better inventory management.

Problem Statement:
In the e-commerce industry, managing a diverse range of products can be challenging, especially when dealing with large inventories. Duplicate entries and unorganized categories can lead to inefficiencies. Your task is to create a Python script that processes a list of product-category pairs, eliminates any duplicates, and categorizes the products neatly.

Instructions:

1. Take user input for the number of products to categorize.
2. For each product, accept the product name and its category.
3. Store these inputs in a suitable data structure, ensuring no duplicate product-category pairs.
4. Categorize the products based on their categories and display the organized list.
5. Handle any input errors using try-except blocks.

Hints:

- Consider using a set or a dictionary to store product-category pairs.
- You can use a set to automatically handle duplicate entries.
- Use a loop to iterate through the number of products.
- Utilize string manipulation to process and display the data neatly.
- Implement try-except blocks for handling non-integer inputs for the number of products.

Exercise 2: Social Media Friend Recommendations

Objective:

Develop a Python program that helps a social media platform recommend new friends to users based on their existing friends' networks. The program should access and analyze user friend lists, identify potential friends, and display recommendations.

Problem Statement:

In social media platforms, friend recommendations are essential for enhancing user engagement. These recommendations are often based on mutual friends. Your task is to create a Python script that takes a user's friend list and the friend lists of others, identifies mutual friends, and suggests potential new friends.

Instructions:

1. Take user input for the number of users on the platform.
2. For each user, accept their name and a list of their friends.
3. Store these inputs in a suitable data structure, such as a dictionary with user names as keys and sets of friends as values.
4. Take input for the name of the user for whom you want to generate friend recommendations.
5. Identify potential friends for this user based on mutual friends with their existing friends.
6. Display the list of recommended friends, excluding the user's current friends.
7. Handle any input errors or cases where the user's name is not found using try-except blocks.

Hints:

- Use a dictionary to map each user's name to a set of their friends for efficient access and manipulation.
- Consider using set intersection to find mutual friends.
- Exclude the user's current friends from the recommendation list.
- Implement try-except blocks for handling incorrect user input or non-existent user names.

Click Next

Exercise 3: Inventory Management for a Retail Store

Objective:
Develop a Python program to manage the inventory of products in a retail store. The program should allow adding new products and updating existing product categories, ensuring that each product is unique in the inventory

Problem Statement:
Efficient inventory management is crucial for retail stores to keep track of their products and categories. Your task is to create a Python script that facilitates adding new products and updating product categories in the store's inventory, represented as a set of products.

Instructions:

1. Initialize an empty set to represent the store's inventory.
2. Implement a function to add a new product to the inventory. Each product should have a name and a category.
3. Implement a function to update the category of an existing product.
4. Provide options for the user to add a product, update a product's category, or display the current inventory.
5. Ensure that the inventory does not contain duplicate products.
6. Use try-except blocks to handle cases where the user tries to update a non-existent product or inputs invalid data.

Hints:

- Use a set to store products, where each product is a tuple containing the product name and category.
- When adding a product, check if it already exists in the inventory.
- For updating a product's category, find the product in the set, remove it, and add a new tuple with the updated category.
- Consider using string manipulation to process and display product details.

Exercise 4: Library Book Catalog Cleanup

Objective:
Create a Python program to manage a library book catalog. The program should allow the removal of books from the catalog and ensure the catalog remains accurate and up-to-date.

Problem Statement:
A library needs to keep its book catalog updated by removing outdated or damaged books. Your task is to develop a Python script that facilitates the removal of books from the library's catalog, which is represented as a set.

Instructions:

1. Initialize a set with a predefined list of books. Each book is identified by its title.
2. Implement a function to remove a book from the catalog by its title.
3. Provide options for the user to remove a book or display the current catalog.
4. Ensure that attempts to remove books that are not in the catalog are handled gracefully.
5. Use try-except blocks to handle any unexpected errors or invalid inputs from the user.

Hints:

- Use a set to store the book titles for efficient removal and searching.
- When removing a book, check if it exists in the catalog first.
- Consider using string manipulation to process the book titles.
- Implement error handling for cases where the user inputs a book title that doesn't exist in the catalog.

Exercise 5: Event Guest List Management

Objective:
Develop a Python program to manage the guest list for an event. The program should facilitate the removal of guests using the `discard` method and provide an option to clear the entire guest list using the `clear` method.

Problem Statement:
Organizing an event requires efficient management of the guest list, including the ability to remove specific guests or clear the entire list as plans change. Your task is to create a Python script that allows for flexible management of an event's guest list, represented as a set.

Instructions:

1. Initialize a set with a predefined list of guests.
2. Implement a function to remove a specific guest from the list using the `discard` method.
3. Implement a function to clear the entire guest list using the `clear` method.
4. Provide options for the user to remove a specific guest, clear the guest list, or display the current guest list.
5. Ensure that the program handles the removal of non-existent guests gracefully.

Hints:

- Use a set to store guest names for efficient addition and removal.
- The `discard` method removes an element from a set if it is present, but does nothing if the element is not found.
- The `clear` method removes all elements from a set, leaving it empty.
- Consider implementing user-friendly messages for successful removals, attempts to remove non-existent guests, and confirmation of clearing the list.

Exercise 6: Wildlife Reserve Animal Tracking

Objective:

Create a Python program to track animal sightings in different zones of a wildlife reserve. The program should use set operations (union, intersection, and difference) to manage and analyze animal sightings across various zones.

Problem Statement:

Managing wildlife reserves involves monitoring animal movements and populations across different zones. Your task is to develop a Python script that uses sets to record animal sightings in multiple zones and then performs analysis using set operations to understand animal distribution and overlaps between zones.

Instructions:

1. Initialize sets for each zone in the wildlife reserve, each containing a list of animals sighted in that zone.
2. Implement functionality to perform the following set operations:
   a. Union: Find all unique animals sighted across multiple zones.
   b. Intersection: Identify common animals sighted in selected zones.
   c. Difference: Determine animals sighted in one zone but not in another.
3. Provide options for the user to select zones and the type of set operation to perform.
4. Display the results of the chosen set operation in a user-friendly format.
5. Handle any input errors or invalid selections by the user.

Hints:

- Utilize the `union()`, `intersection()`, and `difference()` methods for the respective set operations.
- Consider using dictionary mapping for zone names and their corresponding sets of animals.
- Implement user input validation to ensure correct zone names and operation choices are made.
- Use descriptive print statements to make the output clear and informative.

Exercise 7: Museum Artwork Cataloging

Objective:
Develop a Python program to catalog artworks in a museum. The program should use the `enumerate` function to loop through a set of artworks, providing an indexed list for easy reference.

Problem Statement:
A museum needs to create an indexed catalog of its artworks for inventory and visitor reference purposes. Your task is to write a Python script that takes a set of artworks and displays them with an index, making use of the `enumerate` function to loop through the set.

Instructions:

1. Initialize a set with a predefined list of artwork names.
2. Implement a function to loop through the set of artworks using `enumerate`, displaying each artwork with its corresponding index.
3. Provide an option for the user to display the indexed list of artworks.
4. Handle any input errors or invalid selections by the user.

Hints:

- Use the `enumerate()` function in a loop to generate an index for each artwork in the set.
- Consider using a formatted string to display each indexed artwork in a user-friendly manner.
- Ensure the set of artworks is defined with unique entries to represent each artwork accurately.

Exercise 8: Flight Route Network Analysis

Objective:
Create a Python program to analyze flight routes for an airline company. The program should use advanced set methods (`issubset`, `issuperset`, and `isdisjoint`) to compare different route networks.

Problem Statement:
An airline company is looking to optimize its flight routes and needs to analyze the overlap and uniqueness of its routes compared to competitor airlines. Your task is to write a Python script that compares sets of flight routes using advanced set methods, helping the company understand its network coverage.

Instructions:

1. Initialize sets for each airline, each containing a list of destination codes that the airline flies to.
2. Implement functionality to compare the airline's routes with its competitors using the following set operations:
   a. `issubset`: Check if all routes of one airline are covered by another.
   b. `issuperset`: Determine if one airline covers all routes of another airline.
   c. `isdisjoint`: Find out if two airlines have no common routes.
3. Provide options for the user to select airlines and the type of set operation to perform.
4. Display the results of the chosen set operation in a user-friendly format.
5. Handle any input errors or invalid selections by the user.

Hints:

- Utilize the `issubset()`, `issuperset()`, and `isdisjoint()` methods for respective set operations.
- Consider using dictionary mapping for airline names and their corresponding sets of routes.
- Implement user input validation to ensure correct airline names and operation choices are made.
- Use descriptive print statements to make the output clear and informative.

Exercise 9: Unique Ingredient Cookbook Creation

Objective:
Develop a Python program to create a unique ingredient cookbook from various recipes. The program should use set comprehensions to compile a list of unique ingredients from multiple recipes.

Problem Statement:
A chef is compiling a cookbook and wants to ensure that the book includes a wide variety of ingredients. Your task is to write a Python script that processes a list of recipes, each with its own set of ingredients, and uses set comprehensions to create a unique list of ingredients for the cookbook.

Instructions:

1. Initialize a list of sets, where each set contains the ingredients of a different recipe.
2. Use set comprehension to create a single set that contains all unique ingredients from all recipes.
3. Provide an option for the user to display the unique ingredient list.
4. Enhance the set comprehension to optionally include only ingredients that meet certain criteria (e.g., vegetarian, non-dairy).
5. Handle any input errors or invalid selections by the user.

Hints:

- Utilize set comprehension to combine and filter ingredients from different recipe sets.
- Consider adding a conditional within the set comprehension for filtering based on specific criteria.
- Use a user-friendly format to display the unique list of ingredients.
- Ensure your set comprehension handles the uniqueness of ingredients across all recipes.

Exercise 10:

Objective:
Develop a Python program to manage student club memberships at a university. The program should demonstrate best practices in managing sets, especially avoiding common pitfalls like modifying a set while iterating over it.

Problem Statement:
A university needs an efficient system to manage student club memberships. Your task is to create a Python script that manages club memberships, including adding and removing members, while ensuring you follow best practices and avoid common pitfalls associated with set operations.

Instructions:

1. Initialize a dictionary where each key is a club name and its value is a set of member names.
2. Implement functions to add and remove members from club sets.
3. Provide options for the user to modify club memberships or display current memberships.
4. Implement safeguards to prevent modification of sets while iterating over them.
5. Handle any input errors or cases where the club or member's name is not found using try-except blocks.

Hints:

- Use a dictionary to map club names to sets of member names for easy access and modification.
- When removing members, consider creating a copy of the set or a list of items to be removed to avoid modifying the set during iteration.
- Utilize appropriate error handling techniques to manage non-existent club or member names.
- Ensure your program follows Pythonic best practices for set usage.

Conclusion

- [x] What are Sets? - adding and removing elements
- [x] Set operations: Union, Intersection, difference - symmetric difference
- [x] Craft set comprehension