RIPHAH INTERNATIONAL UNIVERISTY, ISLAMABAD

Assignment #02

Bachelors of Software Engineering - 6TH SEMESTER

Submitted to: Ma'am Kauser Nasreen

Submitted by:

Name	Sap Id	Email
Samiyya Aftab	**46891**	**46891@gmail.com**
Mariam Sohail	**45773**	**45773@gmail.com**
Afrah Abdulrub	**47608**	**47608@gmail.com**
Iqra Yaqoob	**44566**	**44566@gmail.com**
Aiman Shafique	**44560**	**44560@gmail.com**

Topic: Managing Student Records

1. Introduction

This report explains the concept of **sets** and their application in student management systems. Sets, which are fundamental elements in mathematics, can be used in software systems to classify, filter, and organize data efficiently. In this context, sets are used to group students based on various academic criteria such as enrollment, attendance, and exam performance.

2. What is a Set?

A **set** is a collection of distinct elements. In software and data management, sets help group related information for better organization and processing.

Example from Scenario: In the student management system, sets are used to group students who are:

- Enrolled in a specific course
- Have passed the midterm exam
- Have high attendance

Using sets in this way simplifies the process of checking eligibility, generating reports, and tracking performance.

3. Why Use Sets in Student Management?

Using sets provides multiple advantages:

- Easy classification and grouping of students.
- Simplifies searching, filtering, and report generation.
- Helps in making decisions, such as checking student eligibility for exams or awards.
- Supports performance analysis based on set conditions like grades or attendance.

4. Identification of Sets in the Scenario

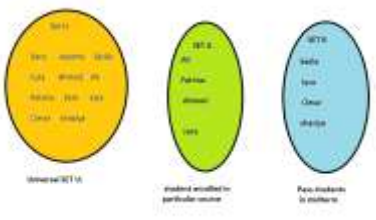
In the presentation, three main sets were identified:

- **Set A**: Students Enrolled in SE2022
  Members: {Sana, Sara, Sadia, Iqra, Sabeen}
- **Set B**: Students Who Passed the Midterm
  Members: {Sara, Ayesha, Sofia}
- **Set C**: Students With ≥75% Attendance
  Members: {Ali, Sara, Usman}

These sets reflect different categories of student data and can be used to perform further operations.

5. Visualizing Student Groups

Using the data above, we can represent sets according there categories



Here **Set A** = (Students who are enrolled particular the course)

Result: {Ali , Fatima, Ahmed , Sara}

Like this **Set B**=(student who pass midterm)

Result:{ Sadia , Iqra, Omar, Shaziya)

This shows that **Sara ,Ali , Fatima, Ahmed** are the student who is enrolled in perticular course and here **Sadia , Iqra, Omar, Shaziya** are studen who pass mid term exams.

Set Operations: Definitions

Set operations are mathematical methods used to compare and combine **sets**. **OR** The set operations are performed on two or more sets to obtain a combination of elements as per the operation performed on them.

1. Union (A ∪ B)

Definition:

The **union** of sets A and B includes **all unique elements** that are in **A**, in **B**, or in **both**.

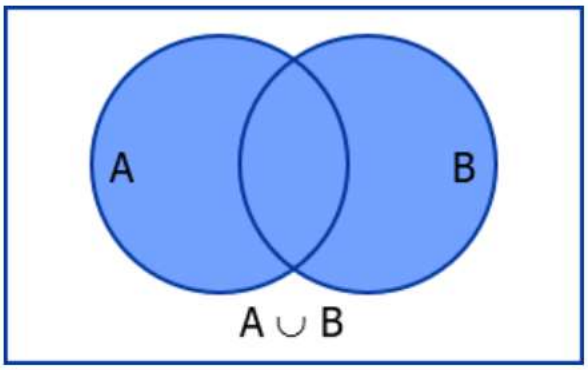
Symbol: A ∪ B

Real Meaning:

Combines two groups into one (no duplicates).

Properties of Union:

- **Commutative:** A ∪ B = B ∪ A
- **Associative:** A ∪ (B ∪ C) = (A ∪ B) ∪ C
- **Idempotent:** A ∪ A = A
- **Identity:** A ∪ ∅ = A



Example:

Let:

A = {Ali, Sara, John}

B = {John, Meena, Ahmed}

Then:

A ∪ B = {Ali, Sara, John, Meena, Ahmed}

Explanation:

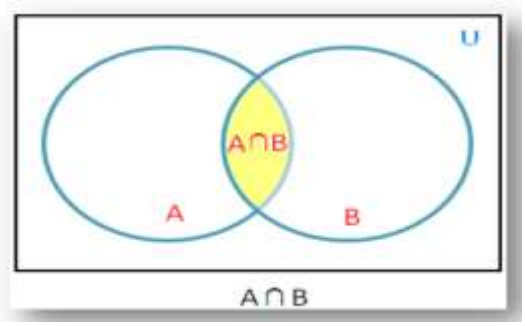
- "John" is in both sets, but appears only once.
- The union brings all unique students from both sets.

2.Intersection (A ∩ B)

Definition:

The **intersection** of sets A and B includes only the elements that are **present in both sets**.

Symbol: A ∩ B



Real Meaning:

Shows the **common elements** shared by both groups.

Properties of Intersection:

- **Commutative:** A ∩ B = B ∩ A
- **Associative:** A ∩ (B ∩ C) = (A ∩ B) ∩ C
- **Idempotent:** A ∩ A = A
- **Identity:** A ∩ U = A (U = universal set)
- **A ∩ ∅ = ∅** (nothing in common with empty set)

Example:

Let:

A = {Ali, Sara, John}

B = {John, Meena, Ahmed}

Then:

A ∩ B = {John}

Explanation:

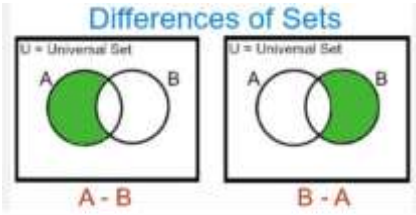
- Only "John" is present in both sets.

3.Difference (A - B)

we explained the **set difference operation**. We focused on how to identify elements that are present in one set (Set A) but not in another (Set B).

What we did:
We provided a clear definition of the operation and used a simple example:

A = {Ali, Sara, John}
B = {John, Meena, Ahmed}
A − B = {Ali, Sara}



Why we did it:
To help understand how this operation works in real-world student data — for instance, finding students who enrolled in one class but not the other.

Reverse Difference (B - A)

In this we focused on the **reverse operation** of the previous slide. Instead of A − B, we now look at B − A.

What we did:
We used the same sets to show that:

B − A = {Meena, Ahmed}
This means that these students are in Set B only, not in Set A.

Why we did it:
To explain the **non-commutative** nature of difference — meaning A − B ≠ B − A. It's important in situations like student transfers or exclusive enrollments.

Properties of Set Difference

This highlighted the mathematical properties of the difference operation.

What we did:
We listed and explained key properties:

Properties of Set Difference

- Not Commutative: A − B ≠ B − A
- Not Associative: (A − B) − C ≠ A − (B − C)
- Identity: A − ∅ = A
- Domination: A − A = ∅

Why we did it:
To show how these rules are applied when designing **logical queries** or managing **student databases**, ensuring correct data filters.

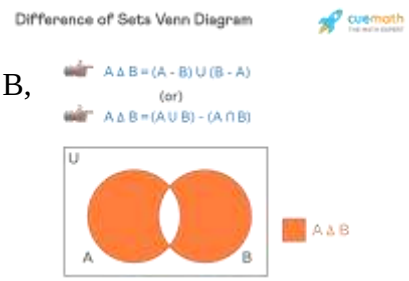
Example of A − B

This revisited the difference operation with a focused **real-life example**.

What we did:
We gave the exact values of A and B again and explained step-by-step how we calculated the result.

Why we did it:
To provide clarity through repetition, ensuring that even complex operations become easy to grasp.

Symmetric Difference (A △ B)



We introduced the symmetric difference — elements that are in A or B,

but not in both.

What we did:
Defined the operation and provided an example:

A = {Areeba, Bilal, Daniyal, Hina, Saad}
B = {Hina, Saad, Komal, Farhan, Zayan}
A △ B = {Areeba, Bilal, Daniyal, Komal, Farhan, Zayan}

Why we did it:
To demonstrate how this operation filters out **common elements** and focuses on **unique ones**, which is very useful in **comparing data** or **tracking changes**.

Implementation in Python

This slide introduced how set operations can be implemented in the **Python programming language**.

What we did:
We showcased Python code snippets that demonstrate the use of sets and how operations like union, intersection, and difference are performed.

Why we did it:
To bridge the gap between **theory and coding**, helping users implement set theory in real applications like **student record systems**

Set Theory in FM (Formal Methods)

What we did:
In this slide, we explained how **Set Theory** plays an important role in **Formal Methods**. Formal Methods (FM) are mathematical techniques used in **software development** to describe and verify systems.

What we explained in detail:

Helps Build Software:
Set theory provides a **solid mathematical foundation** for designing systems logically and structurally. It supports defining states, inputs, outputs, and their relationships.

Manages Data:
Sets help **organize data** — like lists of users, roles, permissions — which is especially useful in **databases and access control** systems.

Clear Rules:
It makes **rules and behaviors** of the software system more precise. By using set-based specifications, we avoid ambiguity.

Why this matters:
This shows that set theory isn't just theoretical — it's directly used to **develop secure, correct, and reliable systems** using mathematical reasoning.

Example we gave:
We showed how sets define **user roles** in a system using terms like:

- `Users = {all registered`
- `ActiveUsers ⊆ Users`
- `AllUsers = Users ∪ ActiveUsers`
- `CommonUsers = Users ∩ ActiveUsers`

This shows **practical use** in defining relationships like who is logged in, who has access, and how they interact.

Benefits of Set Theory in Computing

What we did:
In this slide, we summarized the **key advantages** of using set theory, especially in computer science and software development.

Detailed benefits we explained:

Better Code:
Set-based logic helps write cleaner and more structured code. Developers can handle **groups of data efficiently** using sets instead of complex conditional statements.

Fewer Mistakes:
Since set theory relies on **precise logic**, it helps reduce logical errors in the code. This is extremely valuable when building **sensitive systems** like banking software or security tools.

Reliable Systems:
Systems built using formal set-theoretical methods are more **robust and trustworthy**. They behave as expected because their logic has been **mathematically verified**.

Where it's used:
We mentioned key fields where set theory is applied:

- **Security systems**
- **Databases**
- **Artificial Intelligence (AI)**
  All these areas rely on **managing data and relationships**, which sets are perfect for.

Why this matters:
This slide connects everything we studied to the real world. It proves that set theory is not only useful but essential for building **high-quality, bug-free software systems**.

Implementation:

We implemented three basic set operations—union, intersection, and difference—in multiple languages (C++, Java, JavaScript, R) plus formal Z-notation. Each version follows the same logic:

1. Union collects all unique elements from both sets.
2. Intersection keeps only elements common to both sets.
3. Difference retains elements in the first set that aren't in the second.

For example, given sets A={1,2,3,4} and B={3,4,5,6}, the results are:

- Union: {1,2,3,4,5,6}
- Intersection: {3,4}
- Difference (A–B): {1,2}

Each implementation simply:

- Initializes two sets,

- Applies the built-in or idiomatic operations for union/intersection/difference,
- Prints the result.

Work Division:

Iqra Yaqoob	Real world problem , Documentation
Mariam Sohail	Set definition, key characteristics, Types, scenarios. Documentation
Samiyya Aftab	Set operation(union, intersection , properties), code , Documentation
Aiman Shafique	Set operation(difference , symmetric difference) , implementation(python), Documentation
Afrah Abdulrub	Z Notation ,implementation(c++, java, JavaScript , R)Documentation

GitHub Link:

https://github.com/AfrahAbdulrab/FormalMethods/tree/main