Newton School of Technology

19

Sets in Python

by Gladden Rumao

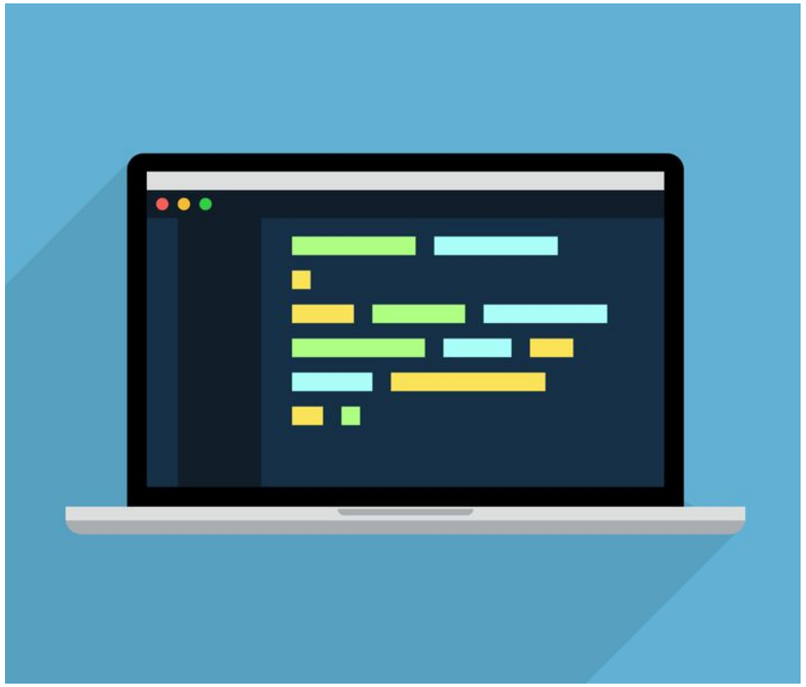
C02: Problem Solving with Programming

Newton School of Technology

Recap of Previous Lecture!

Quick Recap :

- Looping a dictionary
- Dictionary Comprehensions
- Nested Dictionaries



Book Recommendation:

Book Recommendation:

Imagine you are **organizing a book club**. Each month, **different students** will **recommend books** to be read. The final book recommendation list has **no duplicate book recommendations**.

The Club Activities at Rishihood:

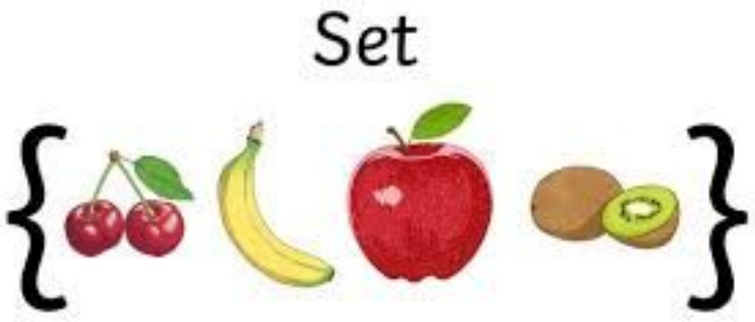
In Python, a set is similar to the final list of book recommendation—it's **a collection of unique elements**. There's **no duplication allowed**—each element is distinct and meaningful.

Newton School of Technology

Introduction to Sets

Definition :

A Set in Python programming is an **unordered built-in collection data type** that is **iterable, mutable** and has **no duplicate elements(unique)**.



Sets in Python :

- Unordered
- Unique
- Sets are defined using **curly braces {}** or the **set()** function.

DO YOU REMEMBER WHAT ELSE USES {}?

Sets in Python :

Creating Sets

Using Curly braces:

```
music_activities = {"singing", "songwriting", "band performance"}
coding_activities = {"hackathon", "algorithm workshop", "debugging bootcamp"}
```

Using the set() Constructor:

```
activities_list = ["Debate Club", "Photography Club", "Coding Bootcamp"]
unique_activities = set(activities_list)
```

Creating Sets

Examples:

```
my_set = {1, 2, 3, 4}
print(my_set)
```

Output: {1, 2, 3, 4}

```
another_set = set([1, 2, 3, 4])
print(another_set)
```

Output: {1, 2, 3, 4}

```
empty_set = set()
print(empty_set)
```

Output: ??

```
tuple_set = set((5, 6, 7, 8))
print(tuple_set)
```

Output: {5, 6, 7, 8}

Creating an Empty Set

To **create an empty set,** you should use the **set() constructor**. Using {} creates an empty dictionary, not an empty set.

```
# Creating an empty set
empty_set = set()
print("Empty Set:", empty_set)  # Output: Empty Set: set()

# Creating an empty dictionary
empty_dict = {}
print("Empty Dictionary:", empty_dict)  # Output: Empty Dictionary: {}
```

Newton School of Technology

Creating Sets

Accessing Set Elements

Newton School of Technology

You can check if an element is present in a set using the '**in**' keyword. Thus, **Sets do not support indexing or slicing** due to its **unordered nature**

```
# Create a set
my_set = {1, 2, 3, 4, 5}

# Check if an element is in the set
print(3 in my_set)  # Output: True
print(6 in my_set)  # Output: False
```

Accessing Set Elements

Iterating through a set: You can iterate over the elements of a set using a **‘for’** loop. Since sets are unordered, the **order of elements** during iteration **may vary.**

```
# Create a set
my_set = {10, 20, 30, 40, 50}

# Iterate through the set
for element in my_set:
    print(element)
```

Output:

10
20
30
40
50

Newton School of Technology

Basic Set Operations

Basic Set Operations

While the **in** keyword is commonly used for **membership testing**, in Python, there are also methods that **allow you to modify sets**.

Adding a Element to Set

.add() : Adds a **single element** to a set. If the element is **already present**, the set **remains unchanged** since sets **do not allow duplicates**.

```
# Create a set
my_set = {1, 2, 3}

# Add an element to the set
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

Adding a Element to Set

```
my_set = {1, 2, 3, 4, 5}

my_set.add(2)
print(my_set)
```

Output:

{1, 2, 3, 4, 5}

(No change since 2 is already in the set)

Updating a Set

.update() : Adds **multiple elements** to the set. Accepts **any iterable** (like a list, set, tuple, etc.) and **adds all elements from the iterable** to the set.

```
set1 = {1, 2, 3, 4}
set2 = {7, 8, 9}

set1.update([5, 6])
print(set1)
set1.update([11, 5, 10], set2)
print(set1)
```

Output:

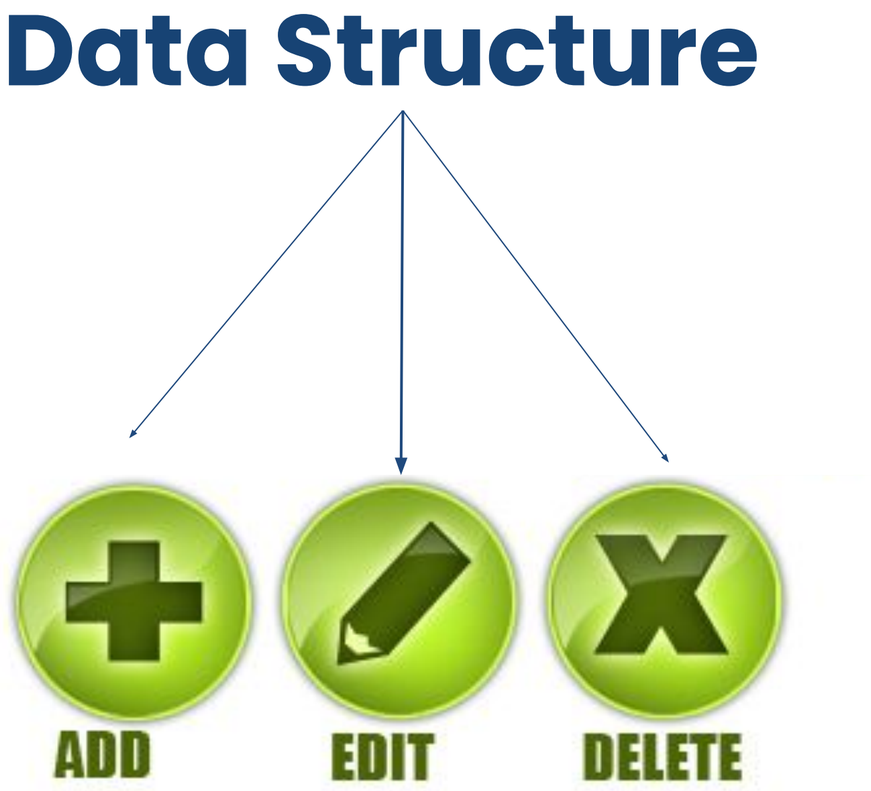
{1, 2, 3, 4, 5, 6}

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Removing Elements:

Newton School of Technology

1. .remove()
2. .discard()
3. .pop()



.REMOVE()

Newton School of Technology

Removes a specific element from a set. If the element is **not present**, it raises a `**KeyError**`

```
# Create a set
my_set = {1, 2, 3, 4}

# Remove an element from the set
my_set.remove(3)
print(my_set)  # Output: {1, 2, 4}
```

Removing an element not in the set

```
try:
    my_set.remove(10)  # This will raise a KeyError
except KeyError as e:
    print(f"Error: {e}")  # Output: Error: 10
```

.DISCARD()

Newton School of Technology

Removes a specific **element** from the **set if it exists**. If the element is not present, it does nothing and **does not raise an error.**

```
my_set = {1, 2, 3, 4}
my_set.discard(2)
print(my_set)
```

Output: {1, 3, 4}

```
my_set.discard(10)
# No error raised
print(my_set)
```

Output: {1, 3, 4}

(No change, 10 was not in the set)

.POP()

Removes and returns an arbitrary element from the set. Since sets are unordered, you **cannot predict which element will be removed.** If the set is **empty**, it raises a `**KeyError**`.

```
my_set = {1, 2, 3, 4, 5}

popped_element = my_set.pop()

print( popped_element)
```

Output:

Clearing a set

The **clear()** method **removes all elements** from the set, effectively emptying it. This method modifies the set **in place** and **does not return any value**.

```
my_set = {1, 2, 3, 4, 5}

print("Initial set:", my_set)

my_set.clear()

print("Set after clearing:", my_set)
```

Output

```
Initial set: {1, 2, 3, 4, 5}
Set after clearing: set()
```

Newton School of Technology

Working with Sets

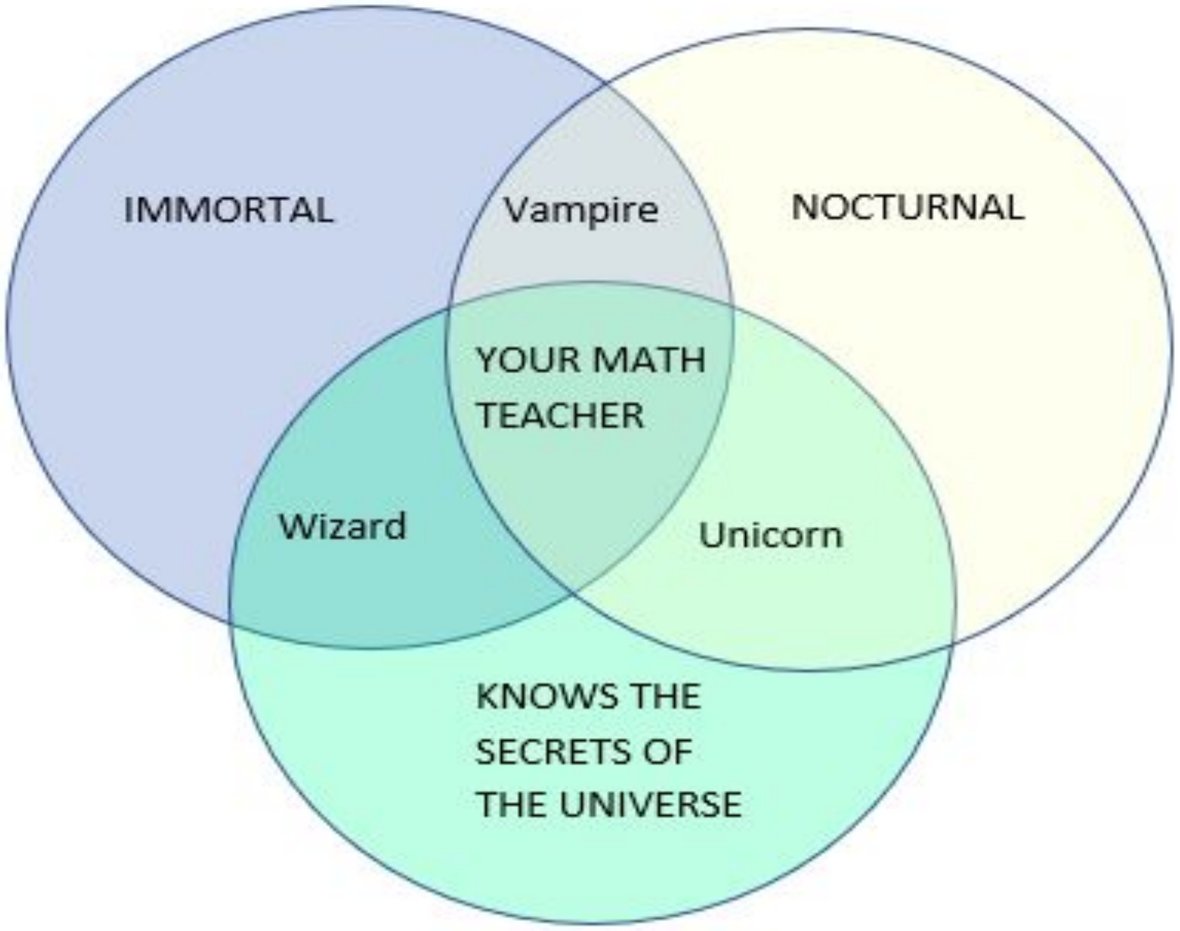
Newton School of Technology

Basic Set Operations

Newton School of Technology

Mathematical Set Operations

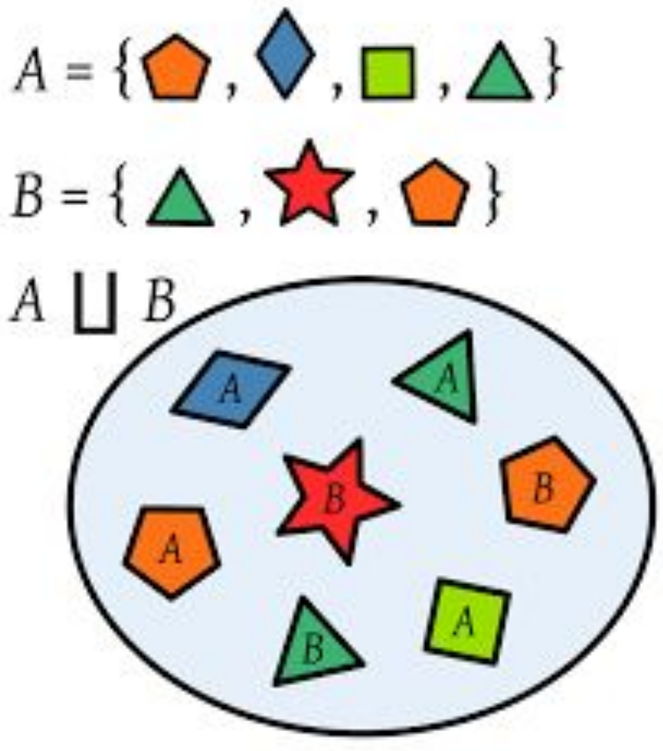
Exploring Types of Set Operations:



Union

Newton School of Technology

Combines **all unique elements from two sets** & returns a **new set** containing all elements that are **in either of the sets.**




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

union_set = set1 | set2
print( union_set)

union_set = set1.union(set2)
print( union_set)
```

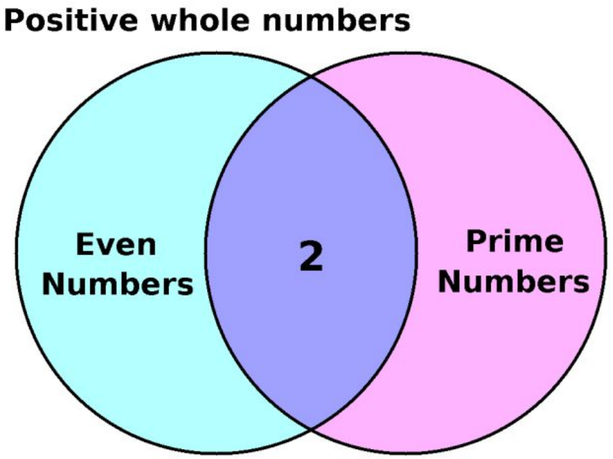
Intersection

Finds **common elements between two sets** & returns **a new set** containing elements that are present in both sets.

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

intersection_set = set1 & set2
print( intersection_set)

intersection_set = set1.intersection(set2)
print(intersection_set)
```



Output:

{3}

Differences

Newton School of Technology

Finds elements that are **in the first set but not in the second set**. The result is a **new set** containing elements that are only in the first set.

```
# Define two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Difference of sets
difference_set = set1 - set2
print("Difference:", difference_set)

# Alternative method
difference_set = set1.difference(set2)
print("Difference (using difference()):", difference_set)
```

Symmetric Differences

Newton School of Technology

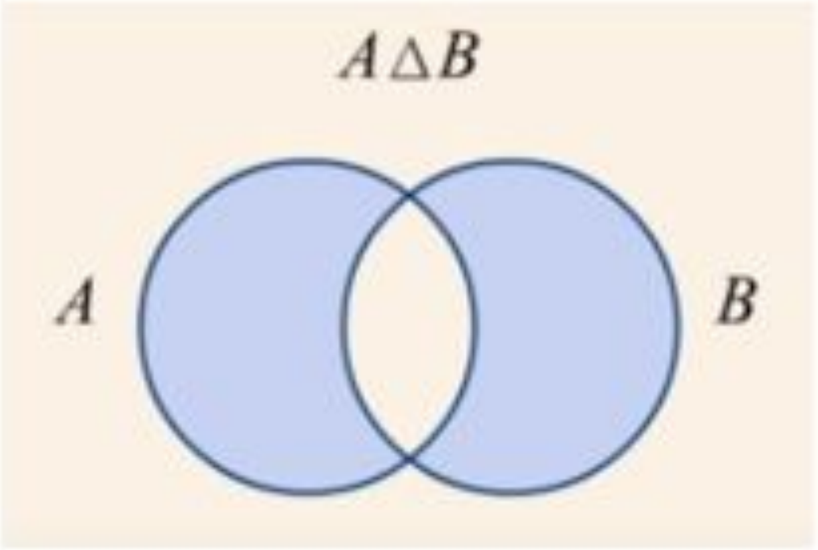
Returns a set containing elements that **are in either of the sets but not in both** or **union of the differences** of the two sets.

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

sym_diff1 = set1 ^ set2
sym_diff2 = set1.symmetric_difference(set2)

print(sym_diff1)
print(sym_diff2)
```

Output: {1, 2, 5, 6}



Newton School of Technology

Union and Intersection

Newton School of Technology

Symmetric & Asymmetric

Newton School of Technology

Other Set Methods

What Will You Do?:

Imagine you have a **set of unique numbers**, and you want to create a copy of this set to **modify it without altering the original set.**

Copying a set:

The **copy()** method creates a **shallow copy** of the set. It creates a **new set** with the **same elements** as the original set, but it is a **different object in memory.**

```
original_set = {1, 2, 3, 4, 5}
copied_set = original_set.copy()

print("Original Set:", original_set)
print("Copied Set:", copied_set)

copied_set.add(6)

print("Original after modification:", original_set)
print("Copied after adding 6:", copied_set)
```

Output

```
Original Set: {1, 2, 3, 4, 5}
Copied Set: {1, 2, 3, 4, 5}
Original after modification: {1, 2, 3, 4, 5}
Copied after adding 6: {1, 2, 3, 4, 5, 6}
```

Newton School of Technology

Set Comprehension

Set Comprehension

Newton School of Technology

Set comprehension is a **concise way to create sets** in Python **using an expression** on **elements** from an **iterable**, optionally **filtered by a condition.**

Syntax:

```
{expression for item in iterable if condition}
```

Example:

```
squared_set = {x*x for x in range(1, 6)}
```

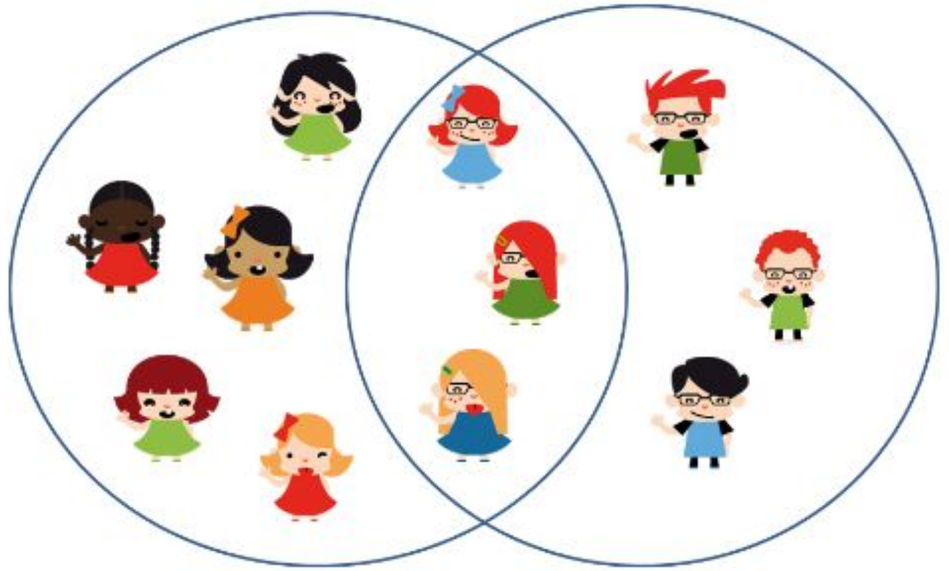
Output: {1, 4, 9, 16, 25}

Newton School of Technology

Set Comprehension

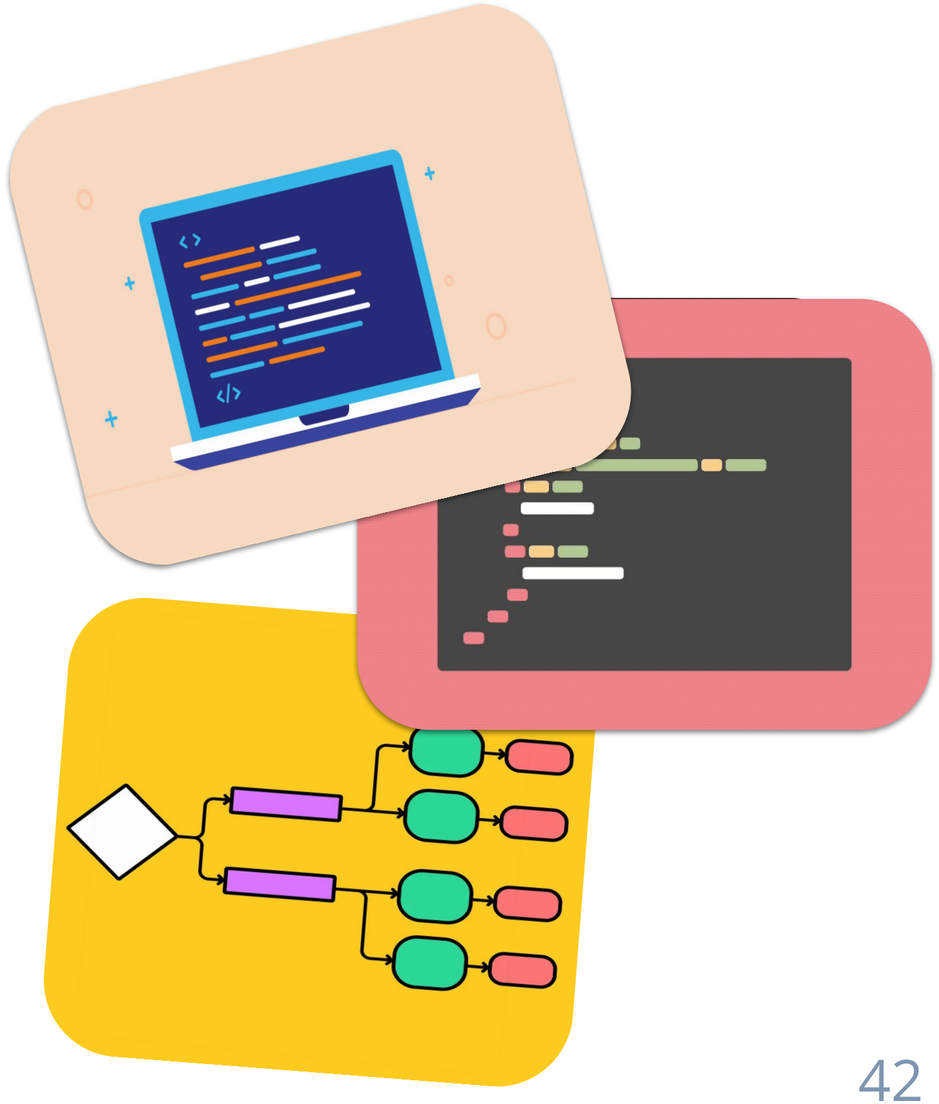
Real-life Use Cases

- Tracking **Unique Visitors** on a Website
- Finding **Common Interests** in Social Networks
- **Managing emails** of Subscribers/ Email Campaign
- **Inventory Management**



Summary

- **Sets:** Unordered, unique elements
- **Creation:** {}, set()
- **Membership Testing**:` in`
- **Access:** Iterate
- **Basic Ops:** add(), remove(), discard(), pop(), clear()
- **Set Ops:** Union |, Intersection &, Difference -, Symmetric Diff ^
- **Methods:** copy(), in
- **Comprehensions:** {x*x for x in range(1, 6)}



Newton School of Technology

Quiz Time!

Thank You!

Activity:

Imagine you and your classmates are planning a **college trip**. Students from different departments—I**nformation Technology (IT), Design, and Psychology**—are signing up for **various activities** during the trip. Your task is to ensure everyone is **registered properly**, and to manage the activities efficiently.

Scenario:

Departments: Information Technology (IT), Design, and Psychology.

Workshops: Hiking, Boating, Camping.

Problem:

- Ensure each student registers only once for an activity.
- Create a master list of all students attending the trip.
- Determine which activities are popular across multiple departments.
- Identify students who are participating in only one activity

Song Playlist at a Party

Song Playlist at a Party:

Imagine you and your friends are planning a party. Each friend has a list of their favorite songs they want to include in the playlist. Naturally, everyone wants to make sure the playlist is diverse and no song is repeated, making it enjoyable for everyone.

Song Playlist at a Party:

In Python, a set is similar to final song playlist—it's **a collection of unique elements**. There’s **no duplication allowed**—each element is distinct and meaningful.