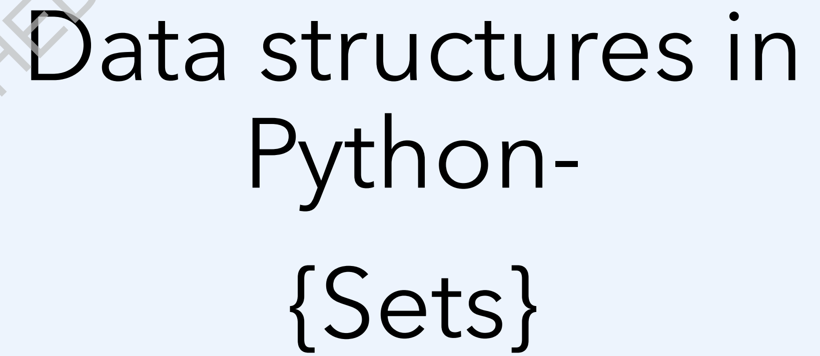


Data structures in Python- {Sets}

By

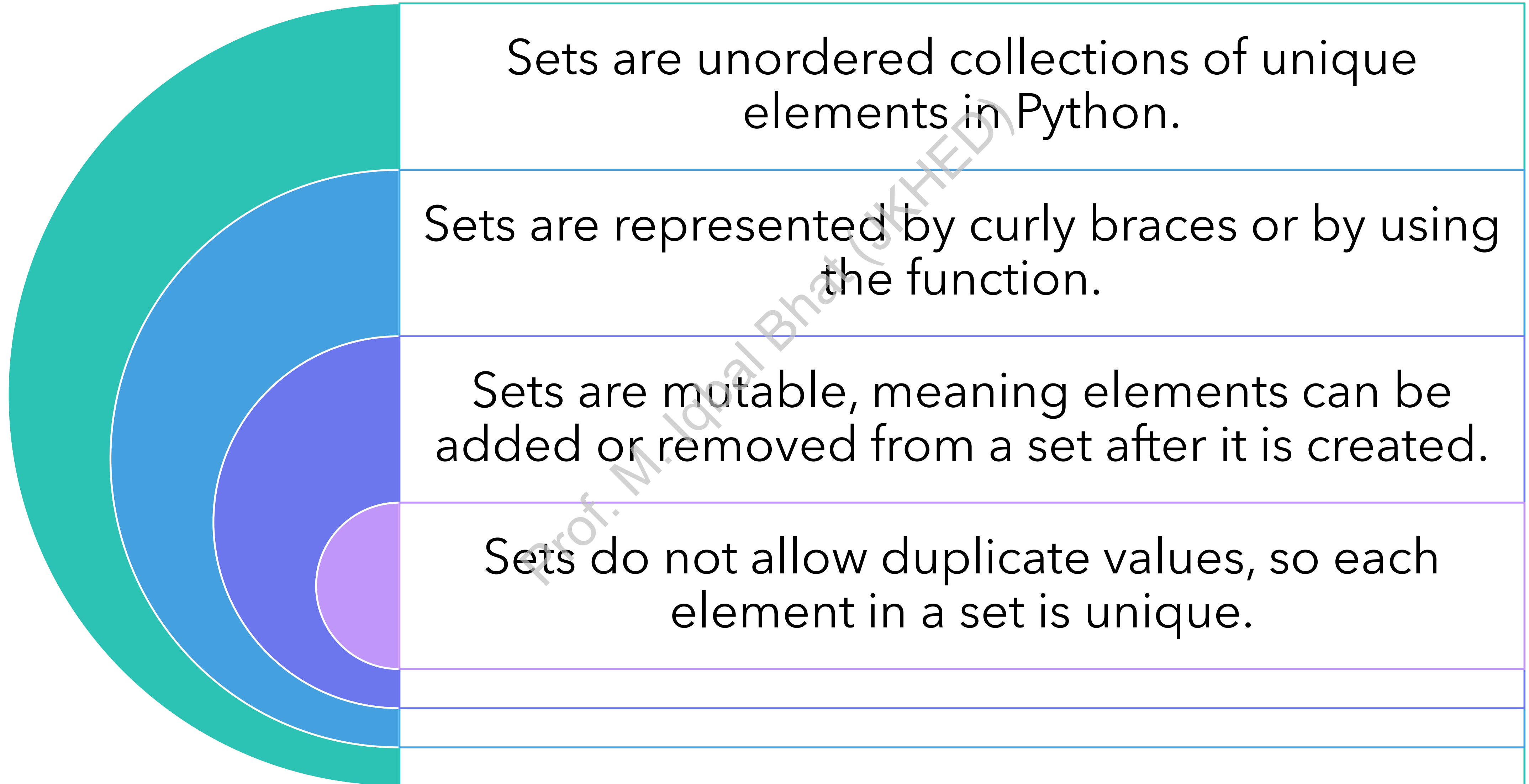
Prof. Muhammad Iqbal Bhat

Department of Higher Education

Government Degree College Beerwah

Prof. M. Iqbal Bhat (JKHED)

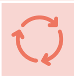
{Sets} in Python:



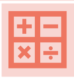
Prof. M. Iqbal Bhat (JKHED)

Creating and modifying {Sets}:

Using curly braces {}: **my_set = {1, 2, 3, 4, 5}**



Using the function: **my_set = set([1, 2, 3, 4, 5])**

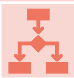


Basic Set Operations:

Adding elements to a set: **my_set.add(6)**

Removing elements from a set: **my_set.remove(3)**

Checking if an element is in a set: **print(4 in my_set) # Output: True**



Checking the length of a set: **print(len(my_set)) # Output: 5**

Prof. M. Iqbal Bhat (JKHED)

Accessing {set} elements:

In Python, you cannot access individual elements of a set using an index like you can with a list or a tuple because sets are unordered collections of unique elements. However, you can iterate over the elements of a set using a loop or convert the set to another data structure like a list or a tuple and access elements using an index.

```
my_set = {1, 2, 3, 4, 5}
for element in my_set:
 print(element)
```

Prof. M. Iqbal Bhat (JKHED)

Common methods for set manipulation

- **union(other_set):** Returns a new set that contains all elements from both sets.
- **intersection(other_set):** Returns a new set that contains only the elements that are common to both sets.
- **difference(other_set):** Returns a new set that contains only the elements that are in the first set but not in the second set.
- **symmetric_difference(other_set):** Returns a new set that contains only the elements that are in either the first set or the second set, but not in both.
- **issubset(other_set):** Returns True if all elements of the set are present in the other set, and False otherwise.
- **issuperset(other_set):** Returns True if all elements of the other set are present in the set, and False otherwise.

Prof. M. Iqbal Bhat (JKHED)

Program examples:

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
# union()
union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5, 6, 7, 8}

# intersection()
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {4, 5}

# difference()
difference_set = set1.difference(set2)
print(difference_set)  # Output: {1, 2, 3}
```

```
# symmetric_difference()
symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)  # Output: {1, 2, 3, 6, 7, 8}

# issubset()
print(set1.issubset(set2))  # Output: False

# issuperset()
print(set1.issuperset(set2))  # Output: False
```

Prof. M. Iqbal Bhat (JKHED)

Uses of {sets}

Prof. M. Iqbal Bhat (JKHED)

1. Removing duplicates: Since sets contain only unique elements, they can be used to remove duplicates from a list or a tuple

```
my_list = [1, 2, 3, 2, 4, 5, 3]
my_set = set(my_list)
print(my_set)  # Output: {1, 2, 3, 4, 5}
```

Prof. M. Iqbal Bhat (JKHED)

2. Membership testing: Sets provide a fast way to check if an element is present in a collection or not.

```
my_set = {1, 2, 3, 4, 5}
if 3 in my_set:
    print("3 is present in the set")
```

Prof. M. Iqbal Bhat (JKHED)

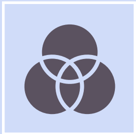
3. Set operations: Sets support various mathematical set operations such as union, intersection, difference, and symmetric difference. These operations can be used to perform operations on sets in a fast and efficient way

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
union_set = set1.union(set2)
intersection_set = set1.intersection(set2)
difference_set = set1.difference(set2)
symmetric_difference_set =
set1.symmetric_difference(set2)
```

Prof. M. Iqbal Bhat (JKHED)

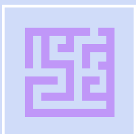
Conclusion:

Sets are useful for eliminating duplicates in a collection. Since sets can only contain unique elements, converting a list or tuple to a set can quickly remove any duplicates.



Sets support various mathematical operations, such as union, intersection, and difference, which can be used to combine or compare sets in a fast and efficient way.

Sets are unordered, which means that elements are not stored in any particular order. However, this allows sets to be very fast for membership testing, since the entire set does not need to be searched to determine if an element is present.



While sets are powerful and versatile, they may not be the right choice for every situation. For example, sets do not allow duplicates, so if you need to store a collection of elements with repeated values, sets may not be the best choice. It's important to consider the specific requirements of your program and choose the appropriate data structure accordingly.

Prof. M. Iqbal Bhat (JKHED)

Prof. M. Iqbal Bhat (JKHED)