**PRACTICAL SET – 05**

**# Set-5 Practical-1 :** Create a set of integers as follows:

* initialize the set directly
* initialize empty set and then add values
* from a list
* from another set
* using range
* update an existing set using another set
* print the elements of set iteratively
* check the functionality of remove and discard

set1 = {

1

,

2

,

3

}

set2 =

set

()

**'''using add func'''**

set2.add(

5

)

set2.add(

6

)

print

(

**"Adding elements in empty set :**

**\**

**n**

**"**

, set2)

list1 = [

10

,

20

,

30

]

**'''using update func to add value from list'''**

set2.update(list1)

print

(

**"**

**\**

**n**

**Updated from another list :**

**\**

**n**

**"**

, set2)

**'''using update func to add value from another set'''**

set3 = {

11

,

22

,

33

}

set2.update(set3)

print

(

**"**

**\**

**n**

**Updated from another set :**

**\**

**n**

**"**

, set2)

**''' print the elements of set iteratively'''**

pr

int

(

**"**

**\**

**n**

**Printing elements of set :**

**\**

**n**

**"**

)

**for**

i

**in**

set2:

print

i

)

(

**'''using remove func'''**

set2.remove(

10

)

print

(

**"Removed**

**\**

**n**

**"**

, set2)

**'''using discard func'''** set2.discard(20)

print(**"Discard\n"**, set2)

**OUTPUT:**

Directly initialized set: {1, 2, 3, 4, 5}

Set after adding values: {8, 6, 7)

Set created from list: (9, 10, 11, 12)

Set created from another set: {13, 14, 15)

Set created from range: 16, 17, 18, 19, 20)

Iterating through the set:

16

17

18

19

20

Set after removing 3: 1, 2, 4, 5}

Set after discarding 4: {1, 2, 5)

Set after discarding non-existent element 10: {1, 2, 5}

**Practical 02:-** Create two sets of integers and find their difference, intersection, union and symmetric difference. Also find subset and superset from these two. Apply methods as well as operators for all operations.

set1 = {1, 2, 3, 4, 5} set2 = {4, 5, 6, 7, 8}

difference\_method = set1.difference(set2)

difference\_operator = set1 - set2

print(f"Difference (set1 - set2)

using method: {difference\_method}")

print(f"Difference (set1 - set2)

using operator: {difference\_operator}")

intersection\_method = set1.intersection(set2)

intersection\_operator = set1 & set2

print(f"Intersection using method: {intersection\_method}")

print(f"Intersection using operator: {intersection\_operator}")

union\_method = set1.union(set2)

union\_operator = set1 | set2

print(f"Union using method: {union\_method}")

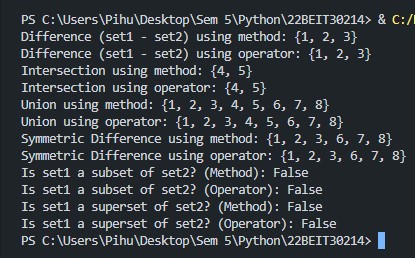
print(f"Union using operator: {union\_operator}")

symmetric\_difference\_method = set1.symmetric\_difference(set2) symmetric\_difference\_operator = set1 ^ set2

print(f"Symmetric Difference using method: {symmetric\_difference\_method}") print(f"Symmetric Difference using operator: {symmetric\_difference\_operator}")

print(f"Is set1 a superset of set2? (Method): {is\_superset\_method}") print(f"Is set1 a superset of set2? (Operator): {is\_superset\_operator}")

**Output:-**



**Practical 03:-** Write a function called find\_dups that takes a list of integers as its input argument and returns a set of those integers that occur two or more times in the list.

Input:-

|  |
| --- |
| def find\_dups(input\_list):  duplicates = set() seen = set() for item in input\_list: if item in seen: duplicates.add(item) else:  seen.add(item) return duplicates  numbers = [1, 2, 3, 4, 5, 2, 3, 6, 7, 8, 9, 3, 10] duplicates = find\_dups(numbers) print(f"Duplicates in the list: {duplicates}") |

**Output:-**

Duplicates in the list: {2, 3}

Practical 04:- The following company details are given for analysis: customer acc no, customer name, purchased product no, product category, unit price. Marketing is interested in understanding customer purchase patterns. Find the answers of following questions:

* How many customers have purchased bread?
* How many customers have purchased butter?
* How many customers have purchased bread and butter?
* Who has purchased bread but not butter?
* Which customers have purchased bread, butter and milk?
* Print the name of the most valuable customers who have purchased all three items.

Input:-

customers = [

{'acc\_no': 101, 'name': 'Alice', 'product\_no': 1, 'category': 'Bread', 'unit\_price': 2.5},

{'acc\_no': 102, 'name': 'Bob', 'product\_no': 2, 'category': 'Butter', 'unit\_price': 3.0},

{'acc\_no': 103, 'name': 'Charlie', 'product\_no': 3, 'category': 'Milk', 'unit\_price': 1.5},

{'acc\_no': 101, 'name': 'Alice', 'product\_no': 2, 'category': 'Butter', 'unit\_price': 3.0},

{'acc\_no': 104, 'name': 'David', 'product\_no': 1, 'category': 'Bread', 'unit\_price': 2.5},

{'acc\_no': 104, 'name': 'David', 'product\_no': 2, 'category': 'Butter', 'unit\_price': 3.0},

{'acc\_no': 105, 'name': 'Eve', 'product\_no': 1, 'category': 'Bread', 'unit\_price': 2.5},

{'acc\_no': 105, 'name': 'Eve', 'product\_no': 3, 'category': 'Milk', 'unit\_price': 1.5},

{'acc\_no': 106, 'name': 'Frank', 'product\_no': 1, 'category': 'Bread', 'unit\_price': 2.5},

{'acc\_no': 106, 'name': 'Frank', 'product\_no': 2, 'category': 'Butter', 'unit\_price': 3.0}, {'acc\_no': 106, 'name': 'Frank', 'product\_no': 3, 'category': 'Milk', 'unit\_price': 1.5},

]

def customers\_purchased\_bread(customers):

bread\_customers = {customer['acc\_no']

for customer in customers

if customer['category'] == 'Bread'}

return len(bread\_customers)

print(f"Customers who purchased bread: {customers\_purchased\_bread(customers)}")

def customers\_purchased\_butter(customers):

butter\_customers = {customer['acc\_no']

for customer in customers

if customer['category']== 'Butter'}

return len(butter\_customers)

print(f"Customers who purchased butter: {customers\_purchased\_butter(customers)}")

def customers\_purchased\_bread\_and\_butter(customers):

bread\_customers = {customer['acc\_no']

for customer in customers

if customer['category'] == 'Bread'}

butter\_customers = {customer['acc\_no'] for customer in customers if customer['category']== 'Butter'}

return len(bread\_customers.intersection(butter\_customers))

print(f"Customers who purchased bread and butter: {customers\_purchased\_bread\_and\_butter(customers)}")

**Output:-**

Customers who purchased bread: 4

Customers who purchased butter: 4

Customers who purchased bread and butter: 3

Customers who purchased bread but not butter: ['Eve', 'Eve']

Customers who purchased bread, butter, and milk: ['Frank', 'Frank', 'Frank']

Most valuable customers who purchased all three items: ['Frank']

**Practical 05:-** Write a program to create an empty tuple, tupple with single value, tuple with multiple values/collections and a tuple with different data types.

Input:-

empty\_tuple = ()

print(f"Empty Tuple: {empty\_tuple}")

single\_value\_tuple = (42,)

print(f"Single Value Tuple: {single\_value\_tuple}")

multiple\_values\_tuple = (1, 2, 3, 4, 5)

print(f"Multiple Values Tuple: {multiple\_values\_tuple}")

mixed\_data\_types\_tuple = (1, "Hello", 3.14, True, [1, 2, 3])

print(f"Mixed Data Types Tuple: {mixed\_data\_types\_tuple}")

**Output:-**

Empty Tuple: ()

Single Value Tuple: (42,)

Multiple Values Tuple: (1, 2, 3, 4, 5)

Mixed Data Types Tuple: (1, 'Hello', 3.14, True, [1, 2, 3])

**Practical 06:-** Check all the methods of tuple.

Input:-

sample\_tuple = (1, 2, 3, 4, 2, 5, 2)

count\_of\_2 = sample\_tuple.count(2)

print(f"Count of 2 in the tuple: {count\_of\_2}")

index\_of\_4 = sample\_tuple.index(4)

print(f"Index of first occurrence of 4: {index\_of\_4}")

**Output:**

Count of 2 in the tuple: 3

Index of first occurrence of 4:3

**Practical 07:-** Write a program to find multiple items of a tuple.

Input:-

sample\_tuple = (10, 20, 30, 40, 50, 60, 70, 80, 90)

items\_by\_index = (sample\_tuple[1], sample\_tuple[3], sample\_tuple[5])

print(f"Items at index 1, 3, and 5: {items\_by\_index}")

items\_by\_slicing = sample\_tuple[2:6]

print(f"Items from index 2 to 5: {items\_by\_slicing}")

items\_with\_step = sample\_tuple[::2]

print(f"Every second item: {items\_with\_step}")

items\_reverse = sample\_tuple[::-1]

print(f"Items in reverse order: {items\_reverse}")

**Output:**

Items at index 1, 3, and 5: (20, 40, 60)

Items from index 2 to 5: (30, 40, 50, 60)

Every second item: (10, 30, 50, 70, 90)

Items in reverse order: (90, 80, 70, 60, 50, 40, 30, 20, 10)

**Practical 08:-** Write a Python script to add a key to a dictionary.

Input:-

my\_dict = { 'name': 'Alice',

'age': 25,

'city': 'New York'

}

my\_dict['profession'] = 'Engineer'

print("Updated Dictionary:", my\_dict)

**Output:-**

Updated Dictionary: ('name': 'Alice', 'age': 25, 'city': 'New York', 'profession': 'Engineer')

**Practical 10:-** Write a Python script to check if a given key already exists in a dictionary.

Input:-

my\_dict = { 'name': 'Alice',

'age': 25,

'city': 'New York'

}

def check\_key\_exists(dictionary, key):

if key in dictionary:

print(f"Key '{key}' exists in the dictionary.")

else:

print(f"Key '{key}' does not exist in the dictionary.")

check\_key\_exists(my\_dict, 'age')

check\_key\_exists(my\_dict, 'profession')

**Output:**

Key 'age' exists in the dictionary.

Key 'profession' does not exist in the dictionary.

**Practical 11:-** Write a Python script to merge two Python dictionaries. Input:-

dict1 = {'name': 'Alice', 'age': 25}

dict2 = {'city': 'New York', 'profession': 'Engineer'}

merged\_dict = dict1.copy()

merged\_dict.update(dict2 )

print("Merged Dictionary using update():",

merged\_dict) merged\_dict\_unpacking = {\*\*dict1, \*\*dict2}

print("Merged Dictionary using unpacking:", merged\_dict\_unpacking)

**Output:**

Merged Dictionary using update(): ['name': 'Alice', 'age': 25, 'city': 'New York', 'profession': 'Engineer'])

Merged Dictionary using unpacking: ('name': 'Alice', 'age': 25, 'city': 'New York', 'profession': 'Engineer')

**Practical 12:-** Write a Python program to remove a key from a dictionary.

Input:-

my\_dict = { 'name': 'Alice',

'age': 25,

'city': 'New York',

'profession': 'Engineer'

}

def remove\_key(dictionary, key):

if key in dictionary:

removed\_value = dictionary.pop(key)

print(f"Key '{key}' removed, Value: {removed\_value}")

else:

print(f"Key '{key}' does not exist in the dictionary.")

remove\_key(my\_dict, 'age') remove\_key(my\_dict, 'salary')

print("Updated Dictionary:", my\_dict)

**Output:**

Key 'age' removed, Value: 25

Key 'salary' does not exist in the dictionary.

Updated Dictionary: {'name': 'Alice', 'city': 'New York', 'profession': 'Engineer')

**Practical 13:-** Write a Python program to create a dictionary from two lists.

Input:-

keys = ['name', 'age', 'city']

values = ['Alice', 25, 'New York']

my\_dict = dict(zip(keys, values)) print("Dictionary created from two lists:")

print(my\_dict)

**Output:**

Dictionary created from two lists:

{'name': 'Alice', 'age': 25, 'city': 'New York'}

**Practical 14:-** Write a Python program to check if all dictionaries in a list are empty or not.

list\_of\_dicts = [{}, {}, {'name': 'Alice'}, {}]

def check\_all\_empty(dicts):

return all(not d for d in dicts)

result = check\_all\_empty(list\_of\_dicts)

if result:

print("All dictionaries in the list are empty.")

else:

print("Not all dictionaries in the list are empty.")

**Output:**

Not all dictionaries in the list are empty.