List Methods

# list is first in first out data structure

# creating List

import copy

lists = [1, 2, 3]

print(lists)

range\_method = range(5)

range2 = list(range(lists))

print(range\_method)

print(range2)

for i in lists:

    print(i)

# creating list using iterables

list\_range = range(5)  # range(0,5)

print(list\_range)

list\_range = list(range(5))  # [0,1,2,3,4]

print(list\_range)

# accessing elements from the list

lists = [1, 2, 'hello', 3]

element = lists[0], lists[1], lists[2]  # but this creates tuples

single\_element = lists[0]      # single element is int type

print(element)  # output : (1, 2, 3)

print(type(element))  # output : tuple

print(single\_element)  # output : 1

print(type(single\_element))  # output : int

# adding elements to a list using append method and extend method

lists.append("hello")

print(lists)

lists.append(2)

lists.extend([7, 'hello'])

print(lists)

# removing last element from a list using pop() method

lists.pop()

print(lists)

# removing specific element from a list using remove() method

lists.remove(2)

print(lists)

lists.clear()

# reversing a list using reverse() method

lists.reverse()

print(lists[::-1])

print(lists)

# sorting a list using sort() method

# lists.sort()  # this is in lists =[7, 'hello', 3, 1]

# print(lists)  # lists.sort()  TypeError: '<' not supported between instances of 'str' and 'int'

lists=[7,'hello',3,1]

lists=[7,3,1]

lists.remove('hello')

print(lists)

# now we can use sort() method

lists.sort()

print(lists)  # Output : [1, 3, 7]

# finding length of a list using len() function

print(len(lists))

lists.append(10)

print(lists)

lists.append(["hi", "bye"])  # it will add as one element

print(lists)

# to add multiple elements at once we need to use extend()

lists.extend(["hi", "bye"])  # each element will be added separately

print(lists)

# to find index of an element in a list

print(lists.index('hi'))  # output : 5

print(lists.index(7))  # output : 2

# this count how many times that number appears in a list

print(lists.count(1))  # output : 1

print(lists.count(10))  # output : 1

# copy a list into another list using copy() method

new\_lists = lists.copy()

print(new\_lists)

reverse = lists.reverse()

# list.copy() → creates a new list and returns it. It does not modify the existing list.

# list.reverse() → reverses the list in place and returns None.

print(reverse)  # returns None because there is no return value for reverse()

print(lists)  # returns reversed list because it changes original list also

# clear all elements from a list using clear() method

lists.clear()

print(lists)

# create a list with same values repeated n times using \* operator

a = [1]\*5

print(a)  # output : [1, 1, 1, 1, 1]

b = ['hi']\*5

print(b)  # output : ['hi', 'hi', 'hi', 'hi', 'hi']

c = [[1]]\*5

print(c)  # output : [[1], [1], [1], [1], [1]]

# this is wrong way to create a list with same values repeated n times because it will create a reference to the same object

# creating like this  [[1]]\*5 means [[1],[1],[1],[1],[1]] which is wrong

# because element inside list like this [1]] is mutable so if you change any element then other elements will also get changed

d = [[1]\*5]\*5

# output : [[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]]

print(d)

e = [[i]\*5 for i in range(5)]

# output : [[0, 0, 0, 0, 0], [1 , 1, 1, 1, 1], [2, 2, 2, 2, 2], [3, 3, 3, 3, 3], [4, 4, 4, 4, 4]]

print(e)

# checks whether 3 is present in the list or not

is\_present = 3 in [1, 2, 3, 4, 5]

print("Check if 3 in list:", is\_present)

max\_val = max([1, 2, 3, 4, 5])   # prints maximum value from the list

min\_val = min([1, 2, 3, 4, 5])

print("Max value:", max\_val, "Min value:", min\_val)

total = sum([1, 2, 3, 4, 5])

print("Sum of list:", total)

# slicing a list

print(lists[:])     # output : [] because lists is empty

print(lists[::-1])   # output : []

lists = list(range(1, 6))

print(lists)         # output : [1, 2, 3, 4, 5]

# output : [1, 3, 5]           because it starts from 0 and goes till end by skipping every second element

print(lists[::2])

# output : [5, 4, 3, 2, 1]   because it starts from -1 and goes till start

print(lists[-1::-1])

# output : []                 it is empty  because it starts from start and goes to last directly not from right side but from left side

print(lists[:-1:-1])

print(lists[::-1])   # output : [5, 4, 3, 2, 1]     printing reversed list

# output : [5, 3, 1]              printing reversed list by skipping every second element

print(lists[::-2])

# concatenating two lists using + operator

lists = list(range(1, 6))

print(lists)          # output : [1, 2, 3, 4, 5]

print(lists+lists)    # output : [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

print(lists\*2)        # output : [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

# checking whether an element exists in a list or not using in keyword

print(1 in lists)       # output : True

print(10 in lists)      # output : False

my\_list = ["apple", "banana", "cherry"]

print("Original list:", my\_list)

shallow\_copy = copy.copy(my\_list)

deep\_copy = copy.deepcopy(my\_list)

print("Shallow copy:", shallow\_copy)

print("Deep copy:", deep\_copy)

print("--------------------")

# iterating over a list using for loop

for i in lists:

    print(i)            # output : 1, 2, 3, 4, 5

print("--------------------")

# iterating over a list using while loop

i = 0

while i < len(lists):

    print(lists[i])     # output : 1, 2, 3, 4, 5

    i += 1

print("--------------------")

# converting string to list using split() method    split() method splits a string into a list

string = "Hello World"

# output : ['Hello', 'World']  #here space is default delimiter

print(string.split())

# output : ['Hello', 'World'] #here space is delimiter means it will split wherever space occurs

print(string.split(" "))

s = "Hello   World"

print(s.split())      # ['Hello', 'World']

# ['Hello', '', '', 'World']  #Here you explicitly tell Python to split only on a single space " ".

print(s.split(" "))

# converting list to string using join() method

print("".join(['Hello', 'World']))  # output : HelloWorld

# output : Hello-World because "-" is used as delimiter here

print("-".join(['Hello', 'World']))

# output : Hello World because " " is used as delimiter here

print(" ".join(['Hello', 'World']))

# output : Hello\nWorld because "\n" prints in next line because \n is newline character

print("\n".join(['Hello', 'World']))

# output : Hello, World because ", " is used as delimiter here

print(", ".join(['Hello', 'World']))

# output : Hello World because " " is used as delimiter he

print("".join(['Hello', ' ', 'World']))

# output : Hello    World because "\t" takes tab space

print("".join(['Hello', '\t', 'World']))

# output : Hello\nWorld because "\n" prints next line

print("".join(['Hello', '\n', 'World']))

# output : Hello\rWorld because "\r"  prints carriage return

print("".join(['Hello', '\r', 'World']))

# output : Hello\fWorld because "\f"  prints form feed means page break

print("".join(['Hello', '\f', 'World']))

# nested list

nested\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(nested\_list)             # output : [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# accessing elements from nested list

print(nested\_list[0][0])       # output : 1

print(nested\_list[1][2])       # output : 6

print(nested\_list[2][1])       # output : 8

# accessing elements from nested list using for loop

for i in nested\_list:

    for j in i:

        print(j)               # output : 1, 2, 3, 4, 5, 6, 7, 8, 9

for i in nested\_list:

    for j in i:

        print(j, end=" ")    # output : 1 2 3 4 5 6 7 8 9

    print()

# accessing elements from nested list using while loop

i = 0

while i < len(nested\_list):

    j = 0

    while j < len(nested\_list[i]):

        print(nested\_list[i][j])   # output : 1, 2, 3, 4, 5, 6, 7, 8, 9

        j += 1

    i += 1

Set Methods

# Python Sets

print("\n--- Creating Sets ---")

my\_set = {1, 2, 3, 4, 5}

print("Create a set:", my\_set)

my\_set2 = set([1, 2, 3, 4, 5])

print("Create set from list:", my\_set2)

empty\_set = set()

print("Empty set:", empty\_set)

print("\n--- Adding / Updating ---")

my\_set.add(6)

print("Add element 6:", my\_set)

my\_set.update([7, 8, 9])

print("Update with multiple elements [7,8,9]:", my\_set)

print("\n--- Removing Elements ---")

my\_set.remove(3)

print("Remove element 3:", my\_set)

my\_set.discard(10)  # no error if element not found

print("Discard element 10 (no error if missing):", my\_set)

element = my\_set.pop()

print("Pop random element:", element, "->", my\_set)

my\_set.clear()

print("Clear set:", my\_set)

print("\n--- Copying ---")

my\_set = {1, 2, 3, 4, 5}

new\_set = my\_set.copy()

print("Copy set:", new\_set)

import copy

shallow\_copy = copy.copy(my\_set)

deep\_copy = copy.deepcopy(my\_set)

print("Shallow copy:", shallow\_copy)

print("Deep copy:", deep\_copy)

print("\n--- Set Operations ---")

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

union\_set = set1.union(set2)

print("Union using union():", union\_set)

union\_set2 = set1 | set2

print("Union using | :", union\_set2)

set1.update(set2)

print("Update set1 with union:", set1)

set1 = {1, 2, 3, 4}

intersect\_set = set1.intersection(set2)

print("Intersection using intersection():", intersect\_set)

intersect\_set2 = set1 & set2

print("Intersection using & :", intersect\_set2)

set1.intersection\_update(set2)

print("Update set1 with intersection:", set1)

set1 = {1, 2, 3, 4}

diff\_set = set1.difference(set2)

print("Difference set1 - set2:", diff\_set)

diff\_set2 = set1 - set2

print("Difference using - :", diff\_set2)

set1.difference\_update(set2)

print("Update set1 with difference:", set1)

set1 = {1, 2, 3, 4}

sym\_diff = set1.symmetric\_difference(set2)

print("Symmetric difference:", sym\_diff)

sym\_diff2 = set1 ^ set2

print("Symmetric difference using ^ :", sym\_diff2)

set1.symmetric\_difference\_update(set2)

print("Update set1 with symmetric difference:", set1)

print("\n--- Subset / Superset Checks ---")

set1 = {1, 2}

set2 = {1, 2, 3, 4}

is\_subset = set1.issubset(set2)

print("Check subset issubset:", is\_subset)

is\_subset2 = set1 <= set2

print("Check subset using <= :", is\_subset2)

is\_proper\_subset = set1 < set2

print("Check proper subset using < :", is\_proper\_subset)

is\_superset = set2.issuperset(set1)

print("Check superset issuperset:", is\_superset)

is\_superset2 = set2 >= set1

print("Check superset using >= :", is\_superset2)

is\_proper\_superset = set2 > set1

print("Check proper superset using > :", is\_proper\_superset)

print("\n--- Disjoint Sets ---")

set3 = {7, 8, 9}

is\_disjoint = set1.isdisjoint(set3)

print("Check if disjoint:", is\_disjoint)

print("\n--- Frozen Sets (Immutable) ---")

frozen = frozenset([1, 2, 3])

print("Frozen set:", frozen)

Dictionary Methods

# Python Dictionaries Methods

from collections import OrderedDict

import copy

print("\n--- Creating Dictionaries ---")

my\_dict = {'a': 1, 'b': 2, 'c': 3}

print("Create dictionary:", my\_dict)

my\_dict2 = dict(a=1, b=2, c=3)

print("Create with dict() constructor:", my\_dict2)

my\_dict3 = dict([('a', 1), ('b', 2), ('c', 3)])

print("Create from list of tuples:", my\_dict3)

print("\n--- Accessing Values ---")

value = my\_dict['a']

print("Access value by key ['a']:", value)

value = my\_dict.get('d', 0)

print("Get value with default (key 'd'):", value)

print("\n--- Adding / Updating ---")

my\_dict['d'] = 4

print("Add/update item d=4:", my\_dict)

my\_dict.update({'e': 5, 'f': 6})

print("Update multiple items:", my\_dict)

print("\n--- Removing Items ---")

del my\_dict['b']

print("Remove item with del ['b']:", my\_dict)

val = my\_dict.pop('c')

print("Pop item ['c']:", val, "->", my\_dict)

val2 = my\_dict.pop('e', None)

print("Pop with default (e):", val2, "->", my\_dict)

item = my\_dict.popitem()

print("Remove and return last item:", item, "->", my\_dict)

my\_dict.clear()

print("Clear dictionary:", my\_dict)

print("\n--- Keys, Values, Items ---")

my\_dict = {'a': 1, 'b': 2, 'c': 3}

print("Reset dict:", my\_dict)

keys = my\_dict.keys()

print("Keys:", keys)

values = my\_dict.values()

print("Values:", values)

items = my\_dict.items()

print("Items:", items)

print("\n--- Copying ---")

new\_dict = my\_dict.copy()

print("Shallow copy using copy():", new\_dict)

shallow\_copy = copy.copy(my\_dict)

deep\_copy = copy.deepcopy(my\_dict)

print("Shallow copy:", shallow\_copy)

print("Deep copy:", deep\_copy)

print("\n--- Membership Tests ---")

exists = 'a' in my\_dict

print("'a' exists:", exists)

not\_exists = 'z' not in my\_dict

print("'z' not exists:", not\_exists)

print("\n--- Dictionary Comprehensions ---")

squared = {x: x\*\*2 for x in range(5)}

print("Squared values:", squared)

even\_squares = {x: x\*\*2 for x in range(10) if x % 2 == 0}

print("Even squares (conditional):", even\_squares)

print("\n--- Merging Dictionaries ---")

dict1 = {'a': 1, 'b': 2}

dict2 = {'c': 3, 'd': 4}

merged1 = dict1 | dict2   # Python 3.9+

print("Merged (| operator, 3.9+):", merged1)

merged2 = {\*\*dict1, \*\*dict2}  # Python 3.5+

print("Merged (\*\* unpacking, 3.5+):", merged2)

print("\n--- Other Useful Methods ---")

length = len(my\_dict)

print("Length of dict:", length)

new\_dict2 = dict.fromkeys(['a', 'b', 'c'], 0)

print("Create dict with default values:", new\_dict2)

my\_dict = {'a': 1, 'b': 2}

val = my\_dict.setdefault('c', 5)

print("Set default for missing key 'c':", val, "->", my\_dict)

val2 = my\_dict.setdefault('b', 10)

print("Set default for existing key 'b':", val2, "->", my\_dict)

ordered = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

print("OrderedDict:", ordered)

Tuple Methods

# Python Tuples

print("\n--- Creating Tuples ---")

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

print("Create a tuple:", my\_tuple)

my\_tuple2 = tuple([1, 2, 3, 4, 5])

print("Create tuple from list:", my\_tuple2)

single\_tuple = (42,)

print("Create tuple with single element:", single\_tuple)

empty\_tuple = ()

print("Empty tuple:", empty\_tuple)

print("\n--- Accessing Elements ---")

element = my\_tuple[0]

print("Access element at index 0:", element)

last = my\_tuple[-1]

print("Negative indexing (last element):", last)

sub\_tuple = my\_tuple[1:4]

print("Slicing [1:4]:", sub\_tuple)

print("\n--- Tuple Operations ---")

new\_tuple = my\_tuple + (6, 7, 8)

print("Concatenate tuples:", new\_tuple)

repeated = my\_tuple \* 3

print("Repeat tuple \*3:", repeated)

a, b, c = (1, 2, 3)

print("Unpack tuple (a, b, c):", a, b, c)

a, \*rest = (1, 2, 3, 4, 5)

print("Unpack with \* (a, \*rest):", a, rest)

a, b = 1, 2

a, b = b, a

print("Swap values:", a, b)

nested = ((1, 2), (3, 4))

print("Nested tuples:", nested)

print("\n--- Tuple Methods ---")

count = my\_tuple.count(2)

print("Count occurrences of 2:", count)

index = my\_tuple.index(3)

print("Index of element 3:", index)

print("\n--- Membership Tests ---")

exists = 3 in my\_tuple

print("Check if 3 exists:", exists)

not\_exists = 10 not in my\_tuple

print("Check if 10 not exists:", not\_exists)

print("\n--- Tuple Properties ---")

length = len(my\_tuple)

print("Length of tuple:", length)

max\_val = max(my\_tuple)

min\_val = min(my\_tuple)

print("Max:", max\_val, "Min:", min\_val)

total = sum(my\_tuple)

print("Sum of tuple:", total)

print("\n--- Sorting ---")

sorted\_tuple = sorted(my\_tuple)

print("Sorted tuple (returns list):", sorted\_tuple)

sorted\_desc = sorted(my\_tuple, reverse=True)

print("Sorted descending:", sorted\_desc)

print("\n--- Conversions ---")

tuple\_from\_list = tuple([1, 2, 3])

print("Convert list to tuple:", tuple\_from\_list)

tuple\_from\_string = tuple("Hello")

print("Convert string to tuple:", tuple\_from\_string)

print("\n--- Advanced Operations ---")

zipped = tuple(zip((1, 2, 3), ('a', 'b', 'c')))

print("Zip tuples:", zipped)

for index, value in enumerate(my\_tuple):

    print(f"Enumerate: index={index}, value={value}")

filtered = tuple(filter(lambda x: x % 2 == 0, my\_tuple))

print("Filter even numbers:", filtered)

mapped = tuple(map(lambda x: x \* 2, my\_tuple))

print("Map function (double each):", mapped)

print("\n--- Boolean Checks ---")

any\_true = any(my\_tuple)

print("Any true in tuple:", any\_true)

all\_true = all(my\_tuple)

print("All true in tuple:", all\_true)

# Create a tuple

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

print(my\_tuple)

# Create tuple from list

my\_tuple = tuple([1, 2, 3, 4, 5])

print(my\_tuple)

# Create tuple with single element

single\_tuple = (42,)

print(single\_tuple)

# Create empty tuple

empty\_tuple = ()

print(empty\_tuple)

# Access element

element = my\_tuple[0]

print("First element:", element)

# Negative indexing

last = my\_tuple[-1]

print("Last element:", last)

# Slicing

sub\_tuple = my\_tuple[1:4]

print("Sliced tuple:", sub\_tuple)

# Concatenate tuples

new\_tuple = my\_tuple + (6, 7, 8)

print("Concatenated:", new\_tuple)

# Repeat tuple

repeated = my\_tuple \* 3

print("Repeated:", repeated)

# Unpack tuple

a, b, c = (1, 2, 3)

print(a, b, c)

# Unpack with \*

a, \*rest = (1, 2, 3, 4, 5)

print("a:", a, "rest:", rest)

# Swap values

a, b = 1, 2

a, b = b, a

print("Swapped:", a, b)

# Nested tuple

nested = ((1, 2), (3, 4))

print(nested)

# Count occurrences

count = my\_tuple.count(2)

print("Count of 2:", count)

# Index of element

index = my\_tuple.index(3)

print("Index of 3:", index)

# Membership check

exists = 3 in my\_tuple

not\_exists = 10 not in my\_tuple

print("Exists 3:", exists, "Not exists 10:", not\_exists)

# Tuple length, min, max, sum

print("Length:", len(my\_tuple))

print("Max:", max(my\_tuple))

print("Min:", min(my\_tuple))

print("Sum:", sum(my\_tuple))

# Sorting

sorted\_tuple = sorted(my\_tuple)

sorted\_desc = sorted(my\_tuple, reverse=True)

print("Sorted:", sorted\_tuple)

print("Sorted desc:", sorted\_desc)

# Convert list to tuple

tuple\_from\_list = tuple([1, 2, 3])

print(tuple\_from\_list)

# Convert string to tuple

tuple\_from\_string = tuple("Hello")

print(tuple\_from\_string)

# Zip tuples

zipped = tuple(zip((1, 2, 3), ('a', 'b', 'c')))

print(zipped)

# Enumerate tuple

for index, value in enumerate(my\_tuple):

    print(index, value)

# Filter tuple

filtered = tuple(filter(lambda x: x % 2 == 0, my\_tuple))

print("Filtered evens:", filtered)

# Map tuple

mapped = tuple(map(lambda x: x \* 2, my\_tuple))

print("Mapped (doubled):", mapped)

# Any / All

print("Any true:", any(my\_tuple))

print("All true:", all(my\_tuple))

String Methods

# Create string

my\_string = "Hello, World!"

print(my\_string)

# Multi-line string

multi\_line = """This is

a multi-line

string"""

print(multi\_line)

# Raw string

raw\_string = r"C:\Users\John"

print(raw\_string)

# f-string

name = "John"

greeting = f"Hello, {name}!"

print(greeting)

# Access char

char = my\_string[1]

print(char)

# String slicing

slice\_ = my\_string[7:12]

print(slice\_)

# Reverse string

reversed\_str = my\_string[::-1]

print(reversed\_str)

# Concatenate

new\_string = "Hello" + " " + "World"

print(new\_string)

# Repeat

repeated = "Ha" \* 3

print(repeated)

# Length

print(len(my\_string))

# Case conversions

print(my\_string.upper())

print(my\_string.lower())

print(my\_string.capitalize())

print(my\_string.title())

print(my\_string.swapcase())

# Strip whitespace/characters

print("   hello   ".strip())

print("   hello   ".lstrip())

print("   hello   ".rstrip())

print("...Hello!!!".strip('!.'))

# Replace

print(my\_string.replace("Hello", "Hi"))

# Split and join

print(my\_string.split(","))

print("a-b-c".split("-"))

print("-".join(["a", "b", "c"]))

# Alignments

print(my\_string.center(20, '\*'))

print(my\_string.ljust(20, '\*'))

print(my\_string.rjust(20, '\*'))

# Tabs

print("Hello\tWorld".expandtabs(8))

# Startswith, endswith

print(my\_string.startswith("Hello"))

print(my\_string.endswith("!"))

# Find, rfind, index

print(my\_string.find("World"))

print(my\_string.rfind("o"))

print(my\_string.index("World"))

# Count

print(my\_string.count("l"))

# String checks

print("abc123".isalnum())

print("abc".isalpha())

print("123".isdigit())

print("Ⅻ".isnumeric())

print("42".isdecimal())

print("hello".islower())

print("HELLO".isupper())

print("Hello".istitle())

print("   ".isspace())

print("hello".isprintable())

print("var\_1".isidentifier())

# Encode / Decode

encoded = my\_string.encode('utf-8')

print(encoded)

decoded = encoded.decode('utf-8')

print(decoded)

# Remove prefix/suffix

print("Hello, World!".removeprefix("Hello, "))

print("Hello, World!".removesuffix("!"))

# Partition / rpartition

print("a,b,c".partition(","))

print("a,b,c".rpartition(","))

# Translate

trans\_table = str.maketrans('aeiou', '12345')

print("hello".translate(trans\_table))

# Format strings

print("Hello, {}!".format("World"))

print("Hello, {name}!".format(name="John"))

print("{0} {1} {0}".format("Hello", "World"))

print("{first} {last}".format({"first": "John", "last": "Doe"}))

# Formatting width / alignment

print("{:<10}".format("left"))

print("{:^10}".format("center"))

print("{:>10}".format("right"))

# Padding, signs, numbers

print("{:05}".format(42))

print("{:+}".format(42))

print("{:.2%}".format(0.12345))

print("{:,}".format(1234567))

# Binary, Octal, Hex, Scientific

print("{:b}".format(42))

print("{:o}".format(42))

print("{:x}".format(42))

print("{:e}".format(1234.5678))

# Zfill

print("42".zfill(5))