List

1. **List Creation**
   - i) `my_list = []` → Empty list
   - ii) `my_list = [1,2,3]` → List with values
   - iii) `my_list = list()` → Empty list using constructor
   - iv) `my_list = list(range(1,6))` → List from range
2. **Indexing and Slicing**
   - i) `my_list[index]` → Access elements at index
   - ii) `my_list[start:stop]` → Slice list from start to stop
   - iii) `my_list[start:stop:step]` → Slice list with step
   - iv) `my_list[-1]` → Last element
   - v) `my_list[::-1]` → Reverse list
3. **List Modification Methods**
   - i) `append(element)` → Add element at the end
   - ii) `extend(iterable)` → Add multiple elements
   - iii) `insert(index, element)` → Insert element at index
   - iv) `remove(element)` → Remove first occurance
   - v) `pop(index)` → Remove and Return occurance at index
   - vi) `clear()` → Remove all elements
4. **Sorting and Reversal Methods**
   - i) `sort()` → Sort list in place
   - ii) `sort(reverse=True)` → Sort list in reverse
   - iii) `reverse()` → Reverse list in place
5. **Searching Methods**
   - i) `index(element)` → Find index of element
   - ii) `count(element)` → count occurences of element
6. **other methods**
   - i) `copy()` → Return shallow copy of list
   - ii) `len()` → Return length of list
7. **List Operations**
   - i) **Arithmatic**
     - a) `list1 + list2` → Concatenate list
     - b) `list * n` → Check Inequality Repeat list n times
   - ii) **Comparision**
     - a) `list1 == list2` → Check if element exists equal
     - b) `list1 != list2` → Check if ~~element~~ Inequal

(iii) Membership
- (a) element in list → Check if element exists
- (b) element not in list → Check if element doesn't exists

(iv) List comprehensions
- (a) [expression for element in iterable] → Create new list
- (b) [expression for element in iterable if condition] → Filter elements

(v) other functions
- (a) all(list) → Check if all elements are True
- (b) any(list) → Check if any element is True
- (c) enumerate(list) → Iterate with index and value
- (d) max(list) → Return max value
- (e) min(list) → Return min value
- (f) sum(list) → Return sum of value
- (g) zip(list1, list2) → Iterate over two list in parallel

Tuple

(1) Tuple Creation
- (i) my_tuple = () → Empty tuple
- (ii) my_tuple = (1,2,3) → Tuple with values
- (iii) my_tuple = tuple() → Using constructor
- (iv) my_tuple = tuple([1,2,3]) → From list
- (v) my_tuple = (1,) → With only single element

(2) Indexing and Slicing
- (i) my_tuple[index] → Access element at index
- (ii) my_tuple[start:stop] → Slice from start to stop
- (iii) my_tuple[start:stop:step] → Slice with steps
- (iv) my_tuple[-1] → Last element
- (v) my_tuple[::-1] → Reverse tuple

(B) Tuple Methods

(1) Tuple Information
- (a) len(my_tuple) → Return length

(2) Conversion
- (a) list(my_tuple) → convert to list.
- (b) set(my_tuple) → convert to set.
- (c) dict(zip(key, my_tuple)) → Convert to dictionary.

(4) Tuple Operation

(i) Arithmatic → Concatenate tuples
  (a) tuple1 + tuple2 →

(ii) Comparision
  (a) tuple1 == tuple2 → Check equality
  (b) tuple1 != tuple2 → Check inequality

(iii) membership
  (a) element in mytuple → Check element is exists
  (b) element not in mytuple → Check element doesn't exist

(iii) Tuple Functions
  (a) all(my-tuple) → Check if all elements are True
  (b) any(my-tuple) → Check if any element is True
  (c) enumerate(my-tuple) → Iterate with index and value
  (d) max(my-tuple) → Return max value
  (e) min(my-tuple) → Return min value
  (f) sum(my-tuple) → Return sum of values
  (g) zip(tuple1, tuple2) → Iterate over two tuples in parallel

(iv) Tuple Unpacking →
  (a) a,b,c = my-tuple → Unpack tuple into variables
  (b) a, b*, c = mytuple → Unpack tuple with arbitrary length.

(5) Key Difference between List and Tuple
  (i) Tuple are Immutable
  (ii) Tuples uses parantheses () while List uses Brackates []
  (iii) Tuples are memory efficient
  (iv) Tuples are suitable for data that should not change.

Set

(1) Set Creation
  (i) my-set = Set() → Empty set.
  (ii) my-set = {1,2,3,4} → Set with values.
  (iii) myset = set([1,2,3]) → Set from List.
  (iv) myset = {1,2,2,3,4,5} → Duplicate values ignored.

② Set Methods

(i) Modification

(a) add(element) → Add element
(b) remove(element) → Remove element
(c) discard(element) → Remove element if exists
(d) pop() → Remove and return arbitrary el.
(e) clear() → Remove all elements

(ii) Operations

(a) union(other_set) → Union of two set
(b) intersection(other_set) → Intersection of two sets
(c) difference(other_set) → Difference of two sets
(d) symmetric_difference(other_set) → Symmetric difference
(e) update(other_set) → Update set with union
(f) intersection_update(other_set) → Update set with intersection
(g) difference_update(other_set) → Update set with difference
(h) symmetric_difference_update(other_set) → Update set with symmetric difference.

(iii) Information

(a) len(my_set) → Returns length of set

③ Set Operations

(i) Arithmetic

(a) set1 | set2 → Union of two sets
(b) set1 & set2 → Intersection of two sets
(c) set1 - set2 → Difference of two sets
(d) set1 ^ set2 → Symmetric difference

(ii) Comparision

(a) set1 == set2 → Check equality
(b) set1 != set2 → Check inequality
(c) set1.issubset(set2) → Check if subset
(d) set1.issuperset(set2) → Check is superset

(iii) Set Function

(a) all(my_set) → Check if all elements are true
(b) any(my_set) → Check if any element is true.

(4) Key characteristics of Set

(i) Unordered collection
(ii) Contains only unique elements
(iii) Mutable
(iv) Suitable for fast membership testing and set operations

(5) Dictionary

(1) Dictionary Creation

(i) my_dict = {} → Empty dictionary
(ii) my_dict = {'a':1, 'b':2} → Dictionary with key value pairs
(iii) my_dict = dict(a=1, b=2, c=3) → Dictionary using dict() constructor
(iv) my_dict = {'a':1, 'b':2, 'b':3} → Duplicate keys ignored (last kept)
(v) my_dict = dict(zip(list1, list2)) → Using 2 list.

(2) Dictionary Methods

(i) Access and Modification

(a) my_dict[key] → Access value by key
(b) my_dict[key] = value → Update or add key value pair
(c) my_dict.get(key, default=None) → Access value by key with default
(d) my_dict.update(other_dict) → Update dictionary with another dictionary

(ii) Deletion

(a) del mydict[key] → Remove key-value pair
(b) my_dict.pop(key) → Remove and return key-value pair
(c) my_dict.popitem() → Remove and return arbitary k-v pair
(d) my_dict.clear() → Remove all key-value pairs.

(iii) Information

(a) len(my_dict) → Return number of key-value pairs
(b) mydict.keys() → Return view object of keys
(c) my_dict.values() → Return view object of values
(d) my_dict.items() → Return view object of key value pair

(3) Dictionary Operations

(i) Arithmatic

(a) my_dict.update(my_dict2) → Update dictionary with other dictionary

(ii) comparison

(a) my_dict1 == my_dict2 → Check equality

(b) my_dict1 != my_dict2 → Check inequality

(iii) Membership

(a) key in my_dict → Check if key exists

(b) key not in my_dict → Check if key doesn't exists

(iv) Dictionary Functions

(a) all(my_dict.values()) → Check all values are True

(b) any(my_dict.values()) → Check if any values is True

(c) max(my_dict.values()) → Return max value

(d) min(my_dict.values()) → Return min value

(e) sum(my_dict.values()) → Return sum of values.

(v) Dictionary comprehension

(a) {key:value for variable in iterable} → Create a new dictionary

(b) {key:value for variable in iterable if condition} → filter a dictionary

(vi) Key Characteristics of Dictionary

(i) Unordered collection

(ii) Contains key-value pairs.

(iii) key must be immutable

(iv) Values can be any type

(v) Dictionaries are mutable

(vi) Suitable for fast lookups and insertions

String

(i) String Creation

(i) my_string = 'Helloworld' → Single quotes / single line

(ii) my_string = "Hello world" → Double quotes / single line

(iii) my_string = """Hello world""" → Triple quotes single quote/ multiline

(iv) my_string = '''Helloworld''' → Triple double quote/ multiline

String Indexing and Slicing

(i) my_str[index] → Access character at index
(ii) my_str[start:stop] → Slicing string from start to stop
(iii) my_str[start:stop:step] → Slice string with steps
(iv) my_str[-1] → Last character
(v) my_str[::-1] → Reverse string

(3) String Methods

(1) Case

(a) lower() → Lowercase conversion
(b) ~~strip~~ upper() → Uppercase conversion
(c) title() → Title case
(d) capitalize() → Capitalize
(e) casefold() → case fold (Remove case distinction)

(2) Formatting

(a) strip() → Remove leading and trailing whitespace
(b) lstrip() → Remove leading whitespaces
(c) rstrip() → Remove trailing whitespaces
(d) center(width) → Center string
(e) ljust(width) → Left justify string
(f) rjust(width) → Right justify string

(3) Searching

(a) find(substring) → Find index of substring
(b) rfind(substring) → Find index of substring from right
(c) index(substring) → Find index of substring from left
(d) count(substring) → Count occurrence of substring

(4) Replacement

(a) replace(old, new) → Replace substring
(b) split(separator) → Split string into list
(c) rsplit(separator) → Split string into list from right
(d) partition(separator) → Partition string into tuple
(e) rpartition(separator) → Partition string into tuple from right

(5) Join

(a) join(iterable) → Join string with separator

(6) Other

(a) encode(encoding) → Encode string to bytes
(b) decode(encoding) → Decode string to bytes to string
(c) format() → Format string using format specifier

(4) String operations

(I) Arithmatic
- (a) + → concat
- (b) * → Repeat

(II) Comparision
- (a) == → Equality check
- (b) != → Inequality check
- (c) > → check greater than
- (d) < → check less than
- (e) >= → check greater than or equal
- (f) <= → check less than or equal

(III) Membership
- (a) in → check if substring exists
- (b) not in → check if substring doesn't exist

(IV) String functions
- (a) len() → Return length of string
- (b) max() → Return maximum character
- (c) min() → Return minimum character
- (d) any() → Check if any character true
- (e) all() → check if all characters are true