Some Methods in Python of: Lists, Tuples, Sets, Dictionary and Strings

Made By @Mohamed Hamed

1st – year Artificial Intelligence student at KFS university, you can find me down:

Facebook

LinkedIn

Table of Contents

Lists:

- ### List Properties:

Property	**Description**
Mutable	Lists can be modified after creation
Ordered	Lists maintain the order of elements
Duplicates	Lists allow duplicate values
Mixed Types	Lists can contain different data types
Dynamic Size	Lists can grow or shrink dynamically

- ### List Methods:

Method	**Description**	**Example**	**Output**
append()	Adds an element to the end	lst.append(5)	[1, 2, 3, 5]
extend()	Extends list with multiple elements	lst.extend([4, 5])	[1, 2, 3, 4, 5]
remove()	Removes first occurrence of an element	lst.remove(2)	[1, 3, 4]
sort()	Sorts list in ascending order	lst.sort()	[1, 2, 3, 4]
pop()	Removes and returns element at index	lst.pop(1)	Element removed: 2
reverse()	Reverses the list	lst.reverse()	[4, 3, 2, 1]
copy()	Creates a shallow copy	b = a.copy()	b == a (True)
clear()	Removes all elements	lst.clear()	[]
count()	Counts occurrences of an element	lst.count(3)	2
index()	Finds index of first occurrence	lst.index(3)	1
insert()	Inserts element at index	lst.insert(2, 99)	[1, 2, 99, 3]
len()	Returns length of the list	len(lst)	4

Tuples:

- **Tuple Properties:**

Property	Description
Immutable	Tuples cannot be modified after creation
Ordered	Tuples maintain their order
Duplicates	Tuples allow duplicate values
Mixed Types	Tuples can contain different data types
Faster than Lists	Tuples are generally faster than lists for iteration

- **Tuple Methods:**

count()

Description: Counts occurrences of a value.

Example: (1,2,2,3).count(2)

Output: 2

index()

Description: Finds the first index of a value.

Example: (1,2,3).index(2)

Output: 1

Concatenation

Description: Joins two tuples

.Example: (1,2) + (3,4)

Output: (1,2,3,4)

Unpacking

Description: Assigns values to variables

.Example: a, b = (1,2)

Output: a=1, b=2

Strings:

- ### String Properties:

Property	Description
Immutable	Strings cannot be changed after creation
Ordered	Characters in a string maintain their order
Indexable	Characters can be accessed by index
Iterable	Strings can be looped over
Unicode Support	Strings support Unicode characters

- ### String Methods:

Method	**Description**	**Example**	**Output**
upper()	Converts to uppercase	'hello'.upper()	'HELLO'
lower()	Converts to lowercase	'HELLO'.lower()	'hello'
replace()	Replaces part of the string	'hello'.replace('l', 'x')	'hexxo'
split()	Splits into a list	'a,b,c'.split(',')	['a', 'b', 'c']
strip()	Removes spaces	' hello '.strip()	'hello'
find()	Finds first occurrence	'hello'.find('l')	2
join()	Joins list into astring	','.join(['a', 'b'])	'a,b'
capitalize()	Capitalizes first letter	'hello'.capitalize()	'Hello'
startswith()	Checks if string starts with	'hello'.startswith('he')	True

endswith()	Checks if string ends with	'hello'.endswith('o')	True
len()	Returns length of the string	len('hello')	5

Comparison of Similar Methods:

Method	List	Tuple	String	Dictionary	Set	Description
len()	✓	✓	✓	✓	✓	Returns the number of elements.
count(value)	✓	✓	✓	✗	✗	Counts occurrences of a value.
index(value)	✓	✓	✓	✗	✗	Returns the first index of a value.
+ (Concatenation)	✓	✓	✓	✗	✗	Combines two sequences.
*** (Repetition)**	✓	✓	✓	✗	✗	Repeats the sequence multiple times.
sorted(iterable)	✓	✓	✓	(keysonly)	✓	Returns a sorted list (does not modify original).
clear()						Removes all elements.
copy()	✓	✓	✗	✓	✓	Creates a shallow copy.

- **Pop()**

Data Type	Method(pop)	Behavior
✓ **List**	`list.pop(index=-1)`	Removes **and returns** the item at `index` (default is last).
✓ **Dictionary**	`dict.pop(key, default)`	Removes **and returns** value for `key` (error if key is missing, unless default is provided).
✓ **Set**	`set.pop()`	Removes **and returns** a **random** item (since sets are unordered).
✗ **Tuple**	`tuple.pop()`	Not supported (Tuples are immutable).
✗ **String**	`string.pop()`	Not supported (Strings are immutable).

---

append(x): adds x as a single element at the end of the list

extend(iterable): adds each element of iterable individually

look down:

```
lst1 = [1, 2, 3]                |       lst1.append([4, 5])

lst2 = [1, 2, 3]                |       lst2.extend([4, 5])

print(lst1)  # Output: [1, 2, 3, [4, 5]] (Nested list inside)

print(lst2)  # Output: [1, 2, 3, 4, 5] (Elements added separately)
```

Sets:

- **Set Properties:**

Property	Description
Unordered	Set elements are **not stored in a specific order** (no guarantee of the order in which elements appear).
Unindexed	Sets **do not support indexing**, meaning you cannot access elements using positions like `set[0]`.
No Slicing	Since sets are unordered and unindexed, **slicing operations** (e.g., `set[1:3]`) **are not possible**.
Immutable Elements Only	A set **can only store immutable objects** (e.g., numbers, strings, tuples), but the set itself is **mutable**.
Unique Elements	Sets **do not allow duplicate values**—each element appears only **once**, even if added multiple times.

- **Set Methods:**

Function	**Description**	**Example**
add()	Adds an element to the set.	`s.add(10)`
remove()	Removes a specific element (raises error if not found).	`s.remove(5)`
discard()	Removes an element if it exists, **no error if not found**.	`s.discard(5)`
pop()	Removes and returns **a random element** (since sets are unordered).	`s.pop()`
clear()	Removes **all elements** from the set.	`s.clear()`
copy()	Returns a **shallow copy** of the set.	`new_set = s.copy()`
union()	Returns a new set with elements from **both sets**.	`s1.union(s2)`
update()	Adds elements from another set (modifies original).	`s1.update(s2)`

issubset()	Returns `True` if a set is **a subset** of another.	`s1.issubset(s2)`
issuperset()	Returns `True` if a set **contains all elements** of another.	`s1.issuperset(s2)`
isdisjoint()	Returns `True` if **sets have no common elements**.	`s1.isdisjoint(s2)`

Dictionary:

- **Dict Properties:**

Property	Description	Example	Output
Unordered	elements in a dict do not have a fixed order.	`d = {'a': 1, 'b': 2}`	.
Mutable	You can change, add, or remove key-value pairs.	`d['c'] = 3`	`{'a': 1, 'b': 2, 'c': 3}`
Keys are unique	A key can only exist once; duplicate keys overwrite previous values.	`d = {'a': 1, 'a': 100}`	`{'a': 100}`
Keys must be immutable	Keys can be strings, numbers, or tuples, but not lists or dictionaries.	`d = { [1,2]: 'value' }`	**Error** (TypeError)
Values can be any type	Values can be any data type (list, dict, tuple, etc.).	`d = {'x': [1, 2, 3]}`	`{'x': [1, 2, 3]}`

- **Dict Methods:**

Method	Description	Example	Output
`dict.keys()`	Returns a view of all keys.	`d = {'a': 1, 'b': 2}; print(d.keys())`	`dict_keys(['a', 'b'])`
`dict.values()`	Returns a view of all values.	`print(d.values())`	`dict_values([1, 2])`
`dict.items()`	Returns key-value pairs as tuples.	`print(d.items())`	`dict_items([('a', 1), ('b', 2)])`
`dict.get(key, default)`	Returns the value for `key`, or `default` if not found.	`print(d.get('a', 0))`	`1`
`dict.update(other_dict)`	Merges another dictionary.	`d.update({'c': 3})`	`{'a': 1, 'b': 2, 'c': 3}`
`dict.pop(key, default)`	Removes key and returns its value.	`print(d.pop('a'))`	`1`

`dict.popitem()`	Removes and returns the last inserted key-value pair.	`print(d.popitem())`	`('b', 2)`
`dict.setdefault(key, default)`	Returns value if key exists; else sets it to `default`.	`print(d.setdefault('c', 10))`	`10`
`dict.clear()`	Removes all items from the dictionary.	`d.clear(); print(d)`	`{}`
`dict.copy()`	Creates a shallow copy of the dictionary.	`d2 = d.copy(); print(d2)`	`{'a': 1, 'b': 2}`
`dict.fromkeys(seq, value)`	Creates a dictionary with keys from `seq`, all set to `value`.	`d = dict.fromkeys(['x', 'y'], 0)`	`{'x': 0, 'y': 0}`