Python for Data Science, AI & Development

Module 2: Python Data Structures

Table of Contents

Dictionaries

A **dictionary** in Python is an **unordered, mutable collection** of key-value pairs. Each key in a dictionary is unique, and it is associated with a value. Dictionaries are optimized for retrieving values when the key is known.

In Python, a dictionary can be created by ***placing a sequence of elements within curly {} braces, separated by a 'comma'.***

Dict = {"key1": 1, "key2": "2", "key3": [3, 3, 3], "key4": (4, 4, 4), ('key5'): 5, (0, 1): 6}

We can also create dictionary ***using dict() constructor***.

```
student = dict(name="John", age=25, major="Computer Science")
print(student)  # Output: {'name': 'John', 'age': 25, 'major': 'Computer Science'}
```

Characteristics

- From Python 3.7 Version onward, Python dictionary is Ordered.
- **Dictionary keys are case sensitive:** the same name but different cases of Key will be treated distinctly.
- **Keys must be immutable:** This means keys can be strings, numbers, or tuples but not lists.
- **Keys must be unique:** Duplicate keys are not allowed and any duplicate key will overwrite the previous value.
- Dictionary internally uses Hashing. Hence, operations like search, insert, delete can be performed in **Constant Time**.

Accessing Dictionary Items

We can access a value from a dictionary by using the **key** within square brackets or **get ()** method.

```
student = {"name": "John", "age": 25}
print(student["name"])  # Output: John
```

```
print(student.get("age"))      # Output: 25
print(student.get("grade"))    # Output: None
print(student.get("grade", "Not Found"))  # Output: Not Found
```

Using ***get ()*** method is safer as it doesn't raise an error if the key is missing.

Modifying a Dictionary

1. Add or Update:

```
student = {"name": "John", "age": 25}
student["age"] = 26  # Update value
student["major"] = "Mathematics"  # Add new key-value pair
print(student)  # Output: {'name': 'John', 'age': 26, 'major': 'Mathematics'}
```

2. Remove Items:

Using del:

```
del student["age"]
print(student)  # Output: {'name': 'John', 'major': 'Mathematics'}
```

Using pop():

```
major = student.pop("major")
print(major)      # Output: Mathematics
print(student)    # Output: {'name': 'John'}
```

Using popitem():

- Removes the last inserted key-value pair (Python 3.7+).

```
item = student.popitem()
print(item)       # Output: ('name', 'John')
print(student)    # Output: {}
```

3. Clear All Items:

Clears all the key-value pairs of dictionaries

```
student.clear()
print(student)  # Output: {}
```

Dictionary Methods

Method	Description
`keys()`	Returns all keys in the dictionary.
`values()`	Returns all values in the dictionary.
`items()`	Returns all key-value pairs as tuples.
`update()`	Updates the dictionary with new key-value pairs.
`copy()`	Creates a shallow copy of the dictionary.
`clear()`	Removes all items from the dictionary.
`pop()`	Removes a key and returns its value.
`popitem()`	Removes the last inserted key-value pair.

```
student = {"name": "John", "age": 25, "major": "CS"}

# Keys, Values, Items
print(student.keys())   # Output: dict_keys(['name', 'age', 'major'])
print(student.values()) # Output: dict_values(['John', 25, 'CS'])
print(student.items())  # Output: dict_items([('name', 'John'), ('age', 25), ('major', 'CS

# Update
new_info = {"age": 26, "grade": "A"}
student.update(new_info)
print(student)  # Output: {'name': 'John', 'age': 26, 'major': 'CS', 'grade': 'A'}
```

Get length of dictionary in Python

Python provides multiple methods to get the length, and we can apply these methods to both simple and nested dictionaries.

```
#Using Len() Function
d ={'Name':'Steve', 'Age':30, 'Designation':'Programmer'}
print(len(d))

#Using List Comprehension
d = {"a": 1, "b": 2, "c": 3}
length = len([key for key in d])
print(length)
```

```python
#Using sum() with 1 for each item
d = {"a": 1, "b": 2, "c": 3}
length = sum(1 for i in d)
print(length)

#Getting length of nested dictionary
d = {
    "person1": {"name": "John", "age": 25},
    "person2": {"name": "Alice", "age": 30},
    "person3": {"name": "Bob", "age": 22}
}
length = len(d)
print(length)
```

Looping Through Dictionaries

1. Looping through keys:

```python
for key in student:
    print(key)
```

gives all the keys of pair.

2. Looping through values:

```python
for value in student.values():
    print(value)
```

gives all the values of pair.

3. Looping through key-value pairs:

```python
for key, value in student.items():
    print(f"{key}: {value}")
```

gives all the key value pairs.

Dictionary Comprehensions

A concise way to create dictionaries.

```python
# Squaring numbers
squares = {x: x**2 for x in range(1, 6)}
print(squares)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Filtering values
filtered = {k: v for k, v in student.items() if k != "age"}
print(filtered)  # Output: {'name': 'John', 'major': 'CS'}
```

Nested Dictionaries

Dictionaries can contain other dictionaries.

```
students = {
    "student1": {"name": "John", "age": 25},
    "student2": {"name": "Alice", "age": 22}
}
print(students["student1"]["name"])  # Output: John
```

Use Cases of Dictionaries

1. Lookup Tables:

Quick retrieval of information.

```
country_codes = {"US": "+1", "IN": "+91", "UK": "+44"}
print(country_codes["IN"])  # Output: +91
```

2. Counting Items:

```
sentence = "hello world hello everyone"
word_count = {}
for word in sentence.split():
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)  # Output: {'hello': 2, 'world': 1, 'everyone': 1}
```

```
from collections import Counter
text = "apple apple banana"
word_count = Counter(text.split())
print(word_count)  # Output: Counter({'apple': 2, 'banana': 1})
```

3. Storing Configurations:

```
config = {"theme": "dark", "version": "1.0", "debug": True}
print(config["theme"])  # Output: dark
```

Summary

Python dictionaries are an essential data structure for handling real-world data efficiently:

- They are flexible, allowing hierarchical and dynamic data storage.
- Operations like search, insert, and delete are fast due to ***hashing.***
- With practical applications ranging from contact books to data analytics, dictionaries are an indispensable tool for Python developers.

Mastering dictionaries will significantly improve your programming efficiency and problem-solving abilities.

Sets

A **set** in Python is a collection of unique, unordered, and unindexed elements. It is a built-in data type designed to hold multiple items in a single variable, ensuring that each element is distinct. Sets are mutable, meaning you can add or remove items, but their elements must be immutable (like integers, strings, or tuples).

Creating a Set in Python

In Python, the most basic and efficient method for creating a set is using curly braces. We can also create set using set() constructor.

```
# Creating sets
set1 = {1, 2, 3, 4}
set2 = set([1, 2, 3, 4, 4])  # Duplicate elements are removed

print("Set 1:", set1)  # Output: {1, 2, 3, 4}
print("Set 2:", set2)  # Output: {1, 2, 3, 4}
```

Python Sets can be created by using the built-in **set()** function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'.

Note: *A Python set cannot have mutable elements like a list or dictionary, as it is immutable.*

Operations on Sets

Basic Operations

- **Adding Elements:** Use add() to add a single element.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

- **Removing Elements:** Use remove() or discard(). The difference is that remove() raises an error if the element is not found, whereas discard() does not.

```
my_set = {1, 2, 3}
my_set.remove(2)  # Removes 2
my_set.discard(4)  # No error even if 4 is not in the set
print(my_set)  # Output: {1, 3}
```

  We can also use pop() to remove elements in a set but it can remove random elements.

- **Clearing and Deleting**

clear(): Removes all elements from the set.
del: Deletes the set entirely.

```
my_set = {1, 2, 3}
my_set.clear()
print(my_set)  # Output: set()
```

Set Operations

1. Union :

Combines all unique elements from two sets.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A | B)  # Output: {1, 2, 3, 4, 5}
print(A.union(B))  # Output: {1, 2, 3, 4, 5}
```

1. Intersection

Finds common elements between sets.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A & B)  # Output: {3}
print(A.intersection(B))  # Output: {3}
```

2. Difference

Elements in one set but not in another.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A - B)  # Output: {1, 2}
print(A.difference(B))  # Output: {1, 2}
```

3. Difference_update

If A and B are two sets. The set **difference ()** method will get the (A – B) and will return a new set. The set **difference_update ()** method modifies the existing set. If (A – B) is performed, then A gets modified into (A – B), and if (B – A) is performed, then B gets modified into (B – A).

4. Symmetric Difference

Elements in either of the sets but not both.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A ^ B)  # Output: {1, 2, 4, 5}
print(A.symmetric_difference(B))  # Output: {1, 2, 4, 5}
```

5. Isdisjoint()

It checks whether the two sets are disjoint or not, if it is disjoint then it returns True otherwise it will return False. Two sets are said to be disjoint when their intersection is null.

```
#Checck if two sets are disjoint if yes True Else False
s1 = {1, 2, 3}
s2 = {4, 5, 6}
print(s1.isdisjoint(s2)) #gives True as there is no common
```

6. Issubset()

This returns True if all elements of a set A are present in another set B which is passed as an argument, and returns False if all elements are not present in Python.

```
#Check Subset
s1 = {1, 2, 3, 4, 5}
s2 = {4, 5}
print(s2.issubset(s1)) #Returns Truue as S2 elements are in S1
```

7. issuperset()

Python Set issuperset() method returns True if all elements of a set B are in set A. Then Set A is the superset of set B.

```
#Check superset
A = {4, 1, 3, 5}
B = {6, 0, 4, 1, 5, 0, 3, 5}

print("A.issuperset(B) : ", A.issuperset(B)) #A is subset not a superset
print("B.issuperset(A) : ", B.issuperset(A)) #True as B has all elements in A
```

Accessing a Set in Python

We can loop through a set to access set items as set is unindexed and do not support accessing elements by indexing. Also we can use in keyword which is membership operator to check if an item exists in a set.

```
my_set = {1, 2, 3, 4}
for element in my_set:
    print(element)
# Output: 1, 2, 3, 4 (order may vary)
```

Sets Methods

Method	Description
`add(element)`	Adds an element to the set.
`clear()`	Removes all elements from the set.
`copy()`	Creates a shallow copy of the set.
`difference()`	Returns the difference of sets.
`difference_update()`	Updates the set with the difference of sets.
`discard(element)`	Removes an element without raising an error.
`intersection()`	Returns the intersection of sets.
`intersection_update()`	Updates the set with the intersection of sets.
`isdisjoint(other_set)`	Checks if sets are disjoint.
`issubset(other_set)`	Checks if the set is a subset of another set.
`issuperset(other_set)`	Checks if the set is a superset of another set.
`pop()`	Removes and returns an arbitrary element.
`remove(element)`	Removes an element, raising an error if it doesn't exist.
`symmetric_difference()`	Returns elements in either set but not in both.
`symmetric_difference_update()`	Updates the set to the symmetric difference.
`union()`	Returns the union of sets.
`update()`	Updates the set with the union of itself and others.

Frozen Sets in Python

A frozenset in Python is a built-in data type that is similar to a set but with one key difference that is immutability. This means that once a frozenset is created, we cannot modify its elements that is we cannot add, remove or change any items in it. Like regular sets, a frozenset cannot contain duplicate elements.

```python
# Creating a frozenset from a list
fset = frozenset([1, 2, 3, 4, 5])
print(fset)

# Creating a frozenset from a set
set1 = {3, 1, 4, 1, 5}
fset = frozenset(set1)
print(fset)
```

```
frozenset({1, 2, 3, 4, 5})
frozenset({1, 3, 4, 5})
```

Key Points to Remember

- Sets are best used when you need to store unique elements.
- They are unordered, so operations involving indexing are not supported.
- Use frozen sets when immutability is required.