



Python Track

DS Basics 2 - Sets and
Dictionaries

Lecture Flow

- Sets
- Dictionaries



Sets



What Is Set?

Set

- A set is a built-in data type in Python that represents an **unordered**, **unindexed** collection of **unique** and **unchangeable** elements.

Initialization

- `set()` constructor

```
empty_set = set() #empty set  
numbers_set = set([1, 2, 3, 4, 5]) # Converts any iterable to a set
```

- Curly braces `{}`

```
numbers_set = {1, 2, 3, 4, 5} # Creates a set with elements  
empty_set = {} # Valid ?
```

What Data Type Can Set Store?

- Sets can store elements of various data types, as long as those elements are **immutable**.
- Integers, Floats, Strings, Tuples, and Booleans.

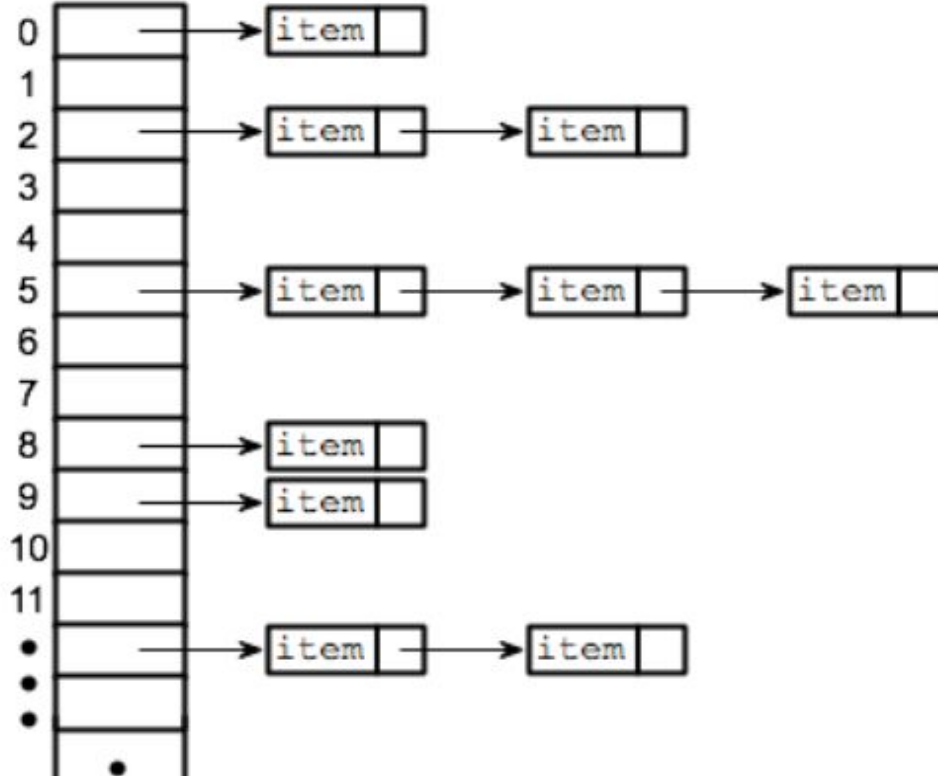
```
my_set = {1, 2.5, "apple", (1, 2, 3), True}
```

```
invalid_set = {1, 2, [3, 4]} # Why invalid?
```


Basics of how Set works

- Sets in Python use a **hash table** to store elements, which is like a large storage box with many compartments (buckets).
- When you add an element to the set, a special function (hash function) **converts the element into a unique code** that determines which bucket the element goes into.
- When you check if an element is in the set (lookup), the hash function is used again to find the compartment where that element should be.

Basics of how Set works



Access Elements

- You cannot access items in a set by referring to an **index or a key**.
- To check if an item is in a set, you use the **in** operator.

```
my_set = {1, 2, 3, 4, 5}
if 3 in my_set:
    #code
```

- You can iterate through all items in a set using a **for-in** loop

```
for item in my_set:
    #code
```

Add And Remove Elements

- The `add()` method is used to add a single element to a set.
- The `update()` method is used to add multiple elements using iterables (lists, tuples)
- The `union()` method Return a set containing the union of sets

```
my_set1 = set()  
my_set2 = {10, 11, 12}
```

```
my_set1.add(1)  
my_set1.add(2)  
my_set1.add(3)
```

```
print(my_set1) # {1, 2, 3}
```

```
my_set1.update([3, 4, 5])  
print(my_set1) # {1, 2, 3, 4, 5}
```

```
my_set3 = my_set1.union(my_set2)  
print(my_set3) # {1, 2, 3, 4, 5, 10, 11, 12}
```

Add And Remove Elements

- The `clear()` method removes all the elements from the set.
- The `remove()` and `discard()` methods remove the specified item.
 - If the item does not exist, `remove()` raises a `KeyError`, while `discard()` does not raise an error.
- `pop()` method removes and returns a **random** element from the set.

```
my_set = {1,2,3,4,5}
my_set.remove(2) # ?
my_set.remove(6) # ?
my_set.discard(2) # ?
my_set.discard(6) # ?
```

```
my_set.pop() # ?
my_set.clear() # ?
```

What Are Valid And Invalid Operators In Set?

Valid Operators

```
set1 = {1, 2, 3}    set2 = {3, 4, 5}
```

- **Union (|)**
union_set = set1 | set2
#returns {1, 2, 3, 4, 5}
- **Intersection (&)**
intersection_set = set1 & set2
#returns {3}
- **Difference (-)**
difference_set = set1 - set2
#returns {1, 2}

Valid Operators

Symmetric Difference (^)

```
symmetric_difference_set = set1 ^ set2  
#returns {1, 2, 4, 5}
```

Subset (<=) and Superset (>=)

```
is_subset = set1 <= set2 #returns False  
is_superset = set1 >= set2 #returns False
```

Equality(==)

```
set1 == set2 #returns False
```


Invalid Operators

- Concatenation (**+**)
- Multiplication (*****)
- Indexing (**[]**) **and** Slicing (**[:]**)

Set Comprehension

- A set comprehension in Python is a concise way to create a set using curly braces {}.

```
Set_name = {expression for item in iterable if condition}
```

```
squares_set = {x**2 for x in range(10)}
```

```
squares_set = {x**2 for x in range(10) if x % 2}
```

Set Methods

Method	Description
<code>add()</code>	Adds an element to the set
<code>clear()</code>	Removes all the elements from the set
<code>copy()</code>	Returns a copy of the set
<code>difference()</code>	Returns a set containing the difference between two or more sets
<code>difference_update()</code>	Removes the items in this set that are also included in another, specified set
<code>discard()</code>	Remove the specified item
<code>intersection()</code>	Returns a set, that is the intersection of two other sets
<code>intersection_update()</code>	Removes the items in this set that are not present in other, specified set(s)
<code>isdisjoint()</code>	Returns whether two sets have a intersection or not
<code>issubset()</code>	Returns whether another set contains this set or not
<code>issuperset()</code>	Returns whether this set contains another set or not
<code>pop()</code>	Removes an element from the set
<code>remove()</code>	Removes the specified element
<code>symmetric_difference()</code>	Returns a set with the symmetric differences of two sets
<code>symmetric_difference_update()</code>	inserts the symmetric differences from this set and another
<code>union()</code>	Return a set containing the union of sets
<code>update()</code>	Update the set with the union of this set and others

Advantage of sets

1. Uniqueness of Elements
2. Fast Membership Testing
3. Mathematical Set Operations

Frozenset

- a frozenset is an **immutable** and **hashable** version of a set.

```
frozenset1 = frozenset([1, 2, 3])  
frozenset2 = frozenset([3, 4, 5])
```

```
frozenset1.add(4) # possible?  
union_frozenset = frozenset1 | frozenset2 # possible?  
intersection_frozenset = frozenset1 & frozenset2 # possible?
```

Exercises

1. [Contains Duplicate - Leetcode](#)
2. [Union of two arrays | Practice | GeeksforGeeks](#)
3. [Contains Duplicate](#)

Dictionaries



What Is Dictionary?

Dictionary

- A dictionary is an **ordered** (as of Python version 3.7) collection of key-value pairs, where each key must be **unique** and is associated with a specific value.
- They are also **mutable** and **dynamic** data structures

Initialization

- `dict()` constructor

```
empty_dictionary = dict() #empty dictionary
dictionary_from_list = dict([('name', 'John'), ('age', 30),
                              ('city', 'New York')])
```

- Curly braces `{}`

```
dictionary = {'name': 'John', 'age': 30, 'city': 'New York'}
```

- Dictionary comprehension

```
sqr_dict = {num: num**2 for num in range(1, 11)}
```

What Can Be Used as Keys in Python Dictionaries?

Valid Data Types for Dictionary Keys

- Dictionary keys must be **immutable** data types to maintain data integrity.
- Integers, Floats, Strings, Tuples and Booleans.

```
my_dictionary = {(1, 2): "Tuple Key", False: "Key"}  
my_dictionary = {[1, 2]: "List Key"} #why invalid?
```

Common Operations In Dictionary

- Access

```
age = my_dict["age"]  
age = my_dict.get("age", 0) #why safe?
```

- Add or Update

```
my_dict["age"] = 20  
my_dict["age"] = my_dict.get("age", 0) + 10
```

- Removing Key

```
value = my_dict.pop("age")
```

Common Operations In Dictionary

- Checking if the key exist

```
if "age" in my_dict
```

- Iterating

- Through Keys:

```
for key in my_dict:
```

- Through key-value pairs:

```
for key, value in my_dict.items():
```

- Through values:

```
for value in my_dict.values():
```

Dictionary Copying

- **Assignment Operator (=):**

```
my_dict1 = {'key1': 'value1', 'key2': 'value2'}  
my_dict2 = my_dict1  
my_dict2['key1'] = 'value3'  
print(my_dict1) # Output?
```

- **Note:** In Python, when you use the assignment operator (=) to assign any object to another variable, both variables reference the same underlying iterable in memory. modifications made to the iterable through one variable will be visible when accessing the iterable through the other variable, and vice versa.

Shallow Copy

- A shallow copy creates a new dictionary but **does not create new copies of the objects** inside the dictionary.
- Changes made to mutable objects (like lists) within the original dictionary will affect the corresponding objects in the copied dictionary, and vice versa.

Shallow Copy

- Shallow copy is performed using methods like
 - `copy()`,
 - `dict()`, or
 - dictionary unpacking (`**original_dict`).

Shallow Copy

```
original_dict = {'key1': ['value1'], 'key2': 'value2'}  
shallow_copied_dict = original_dict.copy()
```

```
original_dict['key1'].append('value3')  
original_dict['key2'] = 'value4'
```

```
print(shallow_copied_dict)  # Output?
```

```
original_dict['key1'] = [1, 2, 3]
```

```
print(shallow_copied_dict)  # Output?
```

Deep Copy

- A deep copy creates a new dictionary and **recursively creates new copies of all objects** inside the original dictionary, including nested objects.
- Changes made to mutable objects within the original dictionary will not affect the corresponding objects in the copied dictionary, and vice versa.

Deep Copy

- Deep copy is performed using the `deepcopy` function from the `copy` module.

Deep Copy

```
import copy

original_dict = {'key1': ['value1'], 'key2': 'value2'}
deep_copied_dict = copy.deepcopy(original_dict)

original_dict['key1'].append('value3')

print(deep_copied_dict)  # Output?
```

Dictionary Methods

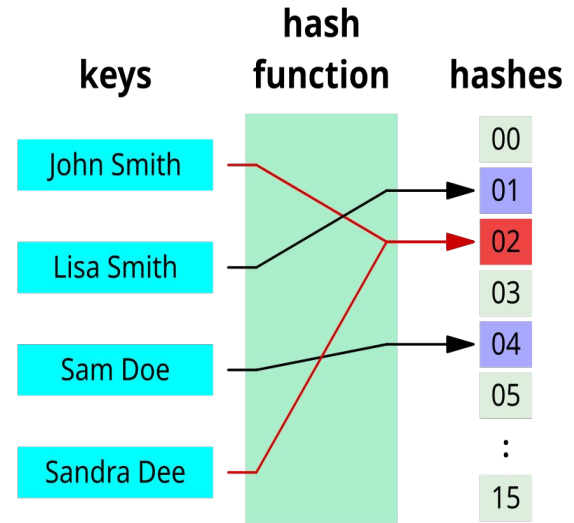
Method	Description
<u>clear</u> ()	Removes all the elements from the dictionary
<u>copy</u> ()	Returns a copy of the dictionary
<u>fromkeys</u> ()	Returns a dictionary with the specified keys and value
<u>get</u> ()	Returns the value of the specified key
<u>items</u> ()	Returns a list containing a tuple for each key value pair
<u>keys</u> ()	Returns a list containing the dictionary's keys
<u>pop</u> ()	Removes the element with the specified key
<u>popitem</u> ()	Removes the last inserted key-value pair
<u>setdefault</u> ()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<u>update</u> ()	Updates the dictionary with the specified key-value pairs
<u>values</u> ()	Returns a list of all the values in the dictionary

Advantage of Dictionaries

1. Efficient Data Retrieval
2. Fast Membership Testing
3. Dynamic Size


Hash Collisions in Sets and Dictionaries

- A **hash collision** occurs when two distinct objects (keys or set elements) have the same hash value.
- This can happen because hash values are finite, so multiple objects can have the same hash code.



Exercises

1. [Missing Number - LeetCode](#)
2. [Find Players With Zero or One Losses - LeetCode](#)
3. [Day 8: Dictionaries and Maps | HackerRank](#)
4. [Check if two arrays are equal or not | Practice | GeeksforGeeks](#)
5. [Array Subset of another array | Practice | GeeksforGeeks](#)



“Success is neither magical nor mysterious. Success is the natural consequence of consistently applying the basic fundamentals.”

- Jim Rohn