

## 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 and unchangeable collection of unique 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?



- It 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 big storage box with lots of compartments (buckets).
- When you add an element to the set, a special function (hash function) turns
  the element into a unique code that determines which bucket the element
  goes into.
- When you want to check if an element is in the set (lookup), the hash function is used again to find the compartment where that element should be.



#### **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_set = set()
my_set.add(1)
my_set.add(2)
my_set.add(2)
my_set.add(3)
print(my_set) # {1,2,3}
my_set.add(3)
```



#### **Add And Remove Elements**

- The clear() method Removes all the elements from the set.
- The remove() and discard() methods Remove the specified item.
- pop() Removes an 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**

```
• Union ( | )
union_set = set1 | set2
```

```
set1 = \{1, 2, 3\}

set2 = \{3, 4, 5\}
```

- Intersection (&)
  intersection\_set = set1 & set2
- Difference (-)
  difference\_set = set1 set2



### **Valid Operators**

```
Symmetric Difference (^)
    symmetric_difference_set = set1 ^ set2

Subset (<=) and Superset (>=)
    is_subset = set1 <= set2
    is_superset = set1 >= set2

Equality(==)
    set1 == set2
```



### **Invalid Operators**

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



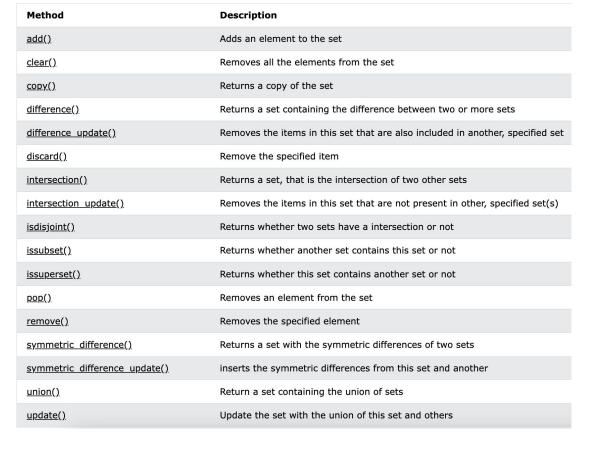
## **Set Comprehension**

• 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





## 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.

```
frozen_set = frozenset([1, 2, 3])
frozen_set = frozenset([1, 2, 3])

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



#### **Exercises**

- 1. Check if All the Integers in a Range Are Covered LeetCode
- 2. <u>Union of two arrays | Practice | GeeksforGeeks</u>
- 3. Check if two arrays are equal or not | Practice | GeeksforGeeks
- 4. <u>Array Subset of another array | Practice | GeeksforGeeks</u>



## **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 data types can be used as dictionary keys in Python?



- 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 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?
```



## **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 is performed using the deepcopy function from the copy module.



## Deep Copy

```
import copy

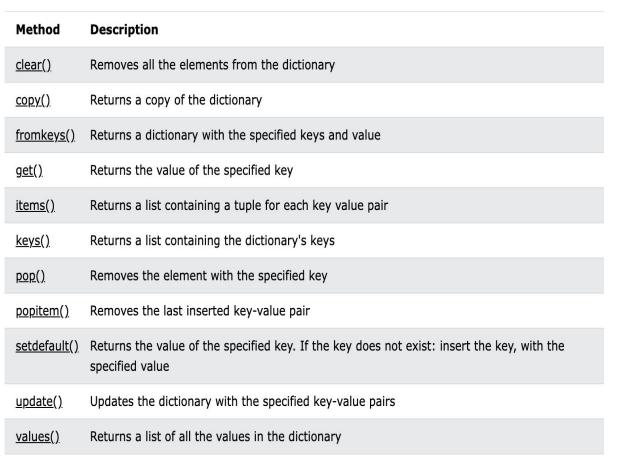
original_dict = {'key1': ['value1'], 'key2': 'value2'}
deep_copied_dict = copy.deepcopy(original_dict)

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

print(deep_copied_dict) # Output?
```



## **Dictionary Methods**





## **Advantage of Dictionaries**

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



### **Exercises**

- 1. <u>Missing Number LeetCode</u>
- 2. Find Players With Zero or One Losses LeetCode



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

- Jim Rohn

