Python Sets - Complete Study Notes

What is a Set?

A **set** is an unordered collection of unique elements in Python. Sets are mutable (can be modified) but contain only immutable (hashable) elements.

Key Characteristics:

- Unordered (no indexing)
- No duplicate elements
- Mutable (can add/remove elements)
- Elements must be immutable/hashable
- Defined using curly braces `{}` or `set()` function

Creating Sets

```
# Empty set
empty_set = set()  # Note: {} creates an empty dict, not set

# Set with elements
fruits = {"apple", "banana", "orange"}
numbers = {1, 2, 3, 4, 5}

# From list (duplicates removed automatically)
my_list = [1, 2, 2, 3, 3, 4]
unique_numbers = set(my_list)  # {1, 2, 3, 4}

# From string
letters = set("hello")  # {'h', 'e', 'l', 'o'}
```

Set Operations

1. Adding Elements

```
fruits = {"apple", "banana"}

# Add single element
fruits.add("orange")
print(fruits)  # {'apple', 'banana', 'orange'}

# Add multiple elements
fruits.update(["mango", "grape"])
fruits.update("kiwi")  # Adds each character: 'k', 'i', 'w', 'i'
print(fruits)  # {'apple', 'banana', 'orange', 'mango', 'grape', 'k', 'i', 'w'}
```

2. Removing Elements

```
fruits = {"apple", "banana", "orange", "mango"}
```

```
# remove() - raises KeyError if element doesn't exist
fruits.remove("banana")

# discard() - doesn't raise error if element doesn't exist
fruits.discard("grape")  # No error even though 'grape' not in set

# pop() - removes and returns arbitrary element
item = fruits.pop()
print(item)  # Could be any element

# clear() - removes all elements
fruits.clear()
print(fruits)  # set()
```

3. Set Mathematical Operations

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Union (|) - all elements from both sets
union_result = set1 | set2  # {1, 2, 3, 4, 5, 6, 7, 8}
union_result = set1.union(set2)  # Same result

# Intersection (&) - common elements
intersection_result = set1 & set2  # {4, 5}
intersection_result = set1.intersection(set2)  # Same result

# Difference (-) - elements in set1 but not in set2
difference_result = set1 - set2  # {1, 2, 3}
difference_result = set1.difference(set2)  # Same result

# Symmetric Difference (^) - elements in either set but not both
sym_diff_result = set1 ^ set2  # {1, 2, 3, 6, 7, 8}
sym_diff_result = set1.symmetric_difference(set2)  # Same result
```

4. Set Comparison Operations

```
set1 = {1, 2, 3}
set2 = {1, 2, 3, 4, 5}
set3 = {1, 2, 3}

# Subset check
print(set1.issubset(set2))  # True
print(set1 <= set2)  # True (same as issubset)

# Proper subset check
print(set1 < set2)  # True

# Superset check
```

```python
print(set2.issuperset(set1))  # True
print(set2 >= set1)  # True (same as issuperset)

# Proper superset check
print(set2 > set1)  # True

# Disjoint check (no common elements)
set4 = {6, 7, 8}
print(set1.isdisjoint(set4))  # True
```

5. Set Membership and Length

```python
fruits = {"apple", "banana", "orange"}

# Membership testing
print("apple" in fruits)  # True
print("grape" not in fruits)  # True

# Length
print(len(fruits))  # 3

# Check if empty
print(bool(fruits))  # True (non-empty)
```

6. Set Iteration

```python
fruits = {"apple", "banana", "orange"}

# Simple iteration
for fruit in fruits:
    print(fruit)

# With enumerate (order not guaranteed)
for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")
```

Frozen Sets

Immutable version of sets - cannot be modified after creation.

```python
# Creating frozen set
frozen_fruits = frozenset(["apple", "banana", "orange"])

# Can be used as dictionary keys (since they're hashable)
fruit_colors = {
    frozenset(["apple", "cherry"]): "red",
    frozenset(["banana"]): "yellow"
}
```

```
# Cannot add/remove elements
# frozen_fruits.add("grape")  # AttributeError
```

Common Use Cases

1. Removing Duplicates

```
# Remove duplicates from list
numbers = [1, 2, 2, 3, 3, 4, 5, 5]
unique_numbers = list(set(numbers))  # [1, 2, 3, 4, 5] (order may vary)
```

2. Finding Common Elements

```
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common = list(set(list1) & set(list2))  # [4, 5]
```

3. Fast Membership Testing

```
# Sets have O(1) average lookup time
large_set = set(range(1000000))
print(999999 in large_set)  # Very fast
```

Tricky Interview Questions

Question 1: Set Creation Gotcha

```
# What's the output?
a = {}
b = set()
print(type(a), type(b))
# Answer: <class 'dict'> <class 'set'>
# {} creates empty dict, not set!
```

Question 2: Mutable Elements

```
# Will this work?
try:
    my_set = {[1, 2], [3, 4]}  # Lists are mutable
except TypeError as e:
    print("Error:", e)
# Answer: Error: unhashable type: 'list'

# This works:
my_set = {(1, 2), (3, 4)}  # Tuples are immutable
```

Question 3: Set Operations Priority

```
# What's the result?
result = {1, 2, 3} | {2, 3, 4} & {3, 4, 5}
```

```python
print(result)
# Answer: {1, 2, 3, 4, 5}
# & has higher precedence than |
# Equivalent to: {1, 2, 3} | ({2, 3, 4} & {3, 4, 5})
# Which is: {1, 2, 3} | {3, 4} = {1, 2, 3, 4}

# To get intersection of union:
result = ({1, 2, 3} | {2, 3, 4}) & {3, 4, 5}
print(result)  # {3, 4}
```

Question 4: Set Modification During Iteration

```python
# What happens here?
my_set = {1, 2, 3, 4, 5}
for item in my_set:
    if item % 2 == 0:
        my_set.remove(item)  # RuntimeError!
# Answer: RuntimeError: Set changed size during iteration

# Correct approach:
my_set = {1, 2, 3, 4, 5}
to_remove = [item for item in my_set if item % 2 == 0]
for item in to_remove:
    my_set.remove(item)
```

Question 5: Set Equality vs Identity

```python
set1 = {1, 2, 3}
set2 = {3, 2, 1}  # Different order
set3 = set1

print(set1 == set2)  # True (same elements)
print(set1 is set2)  # False (different objects)
print(set1 is set3)  # True (same object)
```

Question 6: Nested Sets

```python
# Can you create a set of sets?
try:
    nested = {{1, 2}, {3, 4}}
except TypeError as e:
    print("Error:", e)
# Answer: Error: unhashable type: 'set'

# Use frozensets instead:
nested = {frozenset({1, 2}), frozenset({3, 4})}  # This works!
```

Question 7: Set Comprehension Gotcha

```
# What's the output?
result = {x for x in [1, 1, 2, 2, 3, 3]}
print(result, len(result))
# Answer: {1, 2, 3} 3 (duplicates automatically removed)

# vs list comprehension:
result2 = [x for x in [1, 1, 2, 2, 3, 3]]
print(result2, len(result2))
# Answer: [1, 1, 2, 2, 3, 3] 6
```

Question 8: Performance Question

```
# Which is faster for membership testing?
import time

# Large dataset
data_list = list(range(100000))
data_set = set(data_list)

# Testing membership
target = 99999

# List - O(n)
start = time.time()
result1 = target in data_list
end = time.time()
list_time = end - start

# Set - O(1) average
start = time.time()
result2 = target in data_set
end = time.time()
set_time = end - start

print(f"List time: {list_time}, Set time: {set_time}")
# Set will be significantly faster!
```

Question 9: Set Update Methods

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

# What's the difference?
result1 = set1.union(set2)  # Returns new set
result2 = set1.update(set2)  # Modifies set1 in-place, returns None

print(f"set1 after union: {set1}")    # {1, 2, 3} (unchanged)
print(f"result1: {result1}")          # {1, 2, 3, 4, 5}
print(f"result2: {result2}")          # None
```

Question 10: Boolean Set Operations

```
# True/False are treated as 1/0
bool_set = {True, False, 1, 0}
print(bool_set)  # What's the output?
# Answer: {False, True} or {0, 1}
# True == 1 and False == 0, so duplicates are removed
```

Key Takeaways

1. Sets automatically handle duplicates
2. Elements must be hashable (immutable)
3. Use `set()` for empty set, not `{}`
4. Sets are unordered - no indexing
5. Perfect for membership testing and mathematical operations
6. Be careful when modifying sets during iteration
7. frozenset for immutable sets that can be dict keys
8. Set operations have precedence rules
9. Sets provide O(1) average membership testing