Python Sets Cheat Sheet

Sets and Frozensets in Python

A set is an unordered collection of immutable unique objects.

Introduction to sets

```
# Creating sets
set1 = set()       # empty set
#x = {}            # x is a dictionary, not a set

set2 = {'a', 1, 2, 1, 'a', 2.3, 'a'}                # => {1, 2, 2.3, 'a'} -> unique unordered collection
set3 = set('hellooo python')                        # =>{'n', 'e', 'p', 't', 'o', 'h', 'l', ' ', 'y'}
set4 = set([1, 2.3, 1, 'a', 'a', 2.3, 'b', 5])    # => {1, 2.3, 5, 'a', 'b'}
#set4[0]                                           # TypeError: 'set' object does not support indexing
set5 = {(1, 2), 'a'}     # a set can contain immutable objects like tuples
#set6 = {[1, 2], 'a'}    # TypeError: unhashable type: 'list' -> list is mutable, not allowed in set

s1 = {1, 2, 3}
s2 = {3, 1, 2}
s1 == s2    # => True - order does not matter
s1 is s2    # => False

# The assignment operator (=) creates a reference to the same object
s3 = s1
s3 is s1       # => True
s3 == s1       # => True
s3.add('x')    # adds to the set
print(s1)      # => {1, 2, 3, 'x'}
s3 == s1       # => True
s3 is s1       # => True
```

Iterating over a set

```
some_letters = set('abcabc')
for letter in some_letters:     # prints: c a b
    print(letter, end=' ')
```

Set Membership

```
# in and not in operators test set membership
'a' in some_letters          # => True
'aa' in some_letters         # => False
'bb' not in some_letters     # => True
```

Set Methods

```
# set.copy() creates a copy of the set (not a reference to the same object)
s4 = s1.copy()
s4 is s1     # => False
s4 == s1     # => True
s4.add('z')
s4 == s1     # => False

s1 = {1, 2, 3, 'x'}
# set.pop() removes and returns an arbitrary set element
item = s1.pop()
print(f'item:{item}, s1:{s1}')  # => item:1, s1:{2, 3, 'x'}

# set.discard()  removes an element from a set if it is a member.
# If the element is not a member, do nothing.
s1.discard(2)      # discards element from the set, s1 is {3, 'x'}
s1.discard(22)     # no error if the element doesn't exist

# set.remove() removes an element from a set; it must be a member.
# If the element is not a member, raise a KeyError.
#s1.remove(100)    # KeyError if element doesn't exist
s1.clear()              # Removes all elements from this set
```

Set and Frozenset Operations

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

# set.difference() returns the set of elements that exist only in set1 but not in set2
set1.difference(set2)    # => {1, 2}
```

```
set1 - set2                 # => {1, 2}

# set.symetric_difference() returns the set of elements which are in either of the sets but not in both
set1.symmetric_difference(set2)    # => {1, 2, 4, 5}
set1 ^ set2                        # => {1, 2, 4, 5}

# set.union() returns the set of all unique elements present in all the sets
set1.union(set2)       # => {1, 2, 3, 4, 5}
set1 | set2            # => {1, 2, 3, 4, 5}

# set.intersection() returns the set that contains the elements that exist in both sets
set1.intersection(set2)    # => {3}
set1 & set2                # => {3}

set1.isdisjoint(set2)      # => False
set1.issubset(set2)        # => False
set1 > set2                # => False
set1 <= set2               # => False
{1, 2} <= {1, 2, 3}        # => True

# A frozenset is an immutable set
fs1 = frozenset(set1)
print(fs1)    # => frozenset({1, 2, 3})

# All set methods that don't modify the set are available to frozensets
fs1 & set2    # => frozenset({3})
```

Set Comprehension

General Syntax: set = {expression for item in iterable if condition}

```
set1 = {item for item in [1, 2, 1, 2, 1, 2, 3, 4, 3]}
print(set1)     # => {1, 2, 3, 4}

set2 = {item ** 2 for item in set1 if item %2 == 0}
print(set2)     # => {16, 4}

# Lists with duplicates
cities = ['Paris', 'NYC', 'BERLIN', 'Liverpool', 'Osaka', 'Barcelona']
capitals = ['Paris', 'BERLIN', 'Madrid', 'Paris', 'BERLIN']

# Set comprehension returns a set with capitalized cities in both lists
capitals_unique = {word.capitalize() for word in set(cities) & set(capitals)}
```

```
print(capitals_unique)      # => {'Paris', 'Berlin'}
```