# 6. Dictionaries and Sets

## Objectives

In this chapter, you'll:

- Use dictionaries to represent unordered collections of key–value pairs.
- Use sets to represent unordered collections of unique values.
- Create, initialize and refer to elements of dictionaries and sets.
- Iterate through a dictionary's keys, values and key–value pairs.
- Add, remove and update a dictionary's key–value pairs.

## Objectives (cont.)

- Use dictionary and set comparison operators.
- Combine sets with set operators and methods.
- Use operators `in` and `not in` to determine if a dictionary contains a key or a set contains a value.
- Use the mutable set operations to modify a set's contents.
- Use comprehensions to create dictionaries and sets quickly and conveniently.
- Learn how to build dynamic visualizations and implement more of your own in the exercises.
- Enhance your understanding of mutability and immutability.

## Outline

## Outline (cont.)

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

# 6.1 Introduction

- A **dictionary** is an *unordered* collection which stores **key–value pairs** that map immutable keys to values, just as a conventional dictionary maps words to definitions.
- A **set** is an unordered collection of unique immutable elements.

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.1 Creating a Dictionary

- Create a dictionary by enclosing in curly braces, `{}` , a comma-separated list of key–value pairs, each of the form *key*: *value*.
- Create an empty dictionary with `{}` .

In [1]:

```
country_codes = {'Finland': 'fi', 'South Africa': 'za',
                 'Nepal': 'np'}
```

In [2]:

```
country_codes
```

Out[2]:

```
{'Finland': 'fi', 'South Africa': 'za', 'Nepal': 'np'}
```

- Dictionaries are *unordered* collections.
- Do *not* write code that depends on the order of the key–value pairs.

### Determining if a Dictionary Is Empty

In [3]:

```
len(country_codes)
```

Out[3]:

```
3
```

- Can use a dictionary as a condition to determine if it’s empty—non-empty is `True` and empty is `False`

In [4]:

```
if country_codes:
    print('country_codes is not empty')
else:
    print('country_codes is empty')
```

```
country_codes is not empty
```

In [5]:

```
country_codes.clear()
```

In [6]:

```python
if country_codes:
    print('country_codes is not empty')
else:
    print('country_codes is empty')
```

```
country_codes is empty
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.2 Iterating through a Dictionary

In [1]:

```
days_per_month = {'January': 31, 'February': 28, 'March': 31}
```

In [2]:

```
days_per_month
```

Out[2]:

```
{'January': 31, 'February': 28, 'March': 31}
```

- Dictionary method **`items`** returns each key–value pair as a tuple:

In [3]:

```
for month, days in days_per_month.items():
    print(f'{month} has {days} days')
```

```
January has 31 days
February has 28 days
March has 31 days
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.3 Basic Dictionary Operations

- Intentionally provided the incorrect value `100` for the key `'X'`:

In [1]:

```
roman_numerals = {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
```

In [2]:

```
roman_numerals
```

Out[2]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
```

### Accessing the Value Associated with a Key

In [3]:

```
roman_numerals['V']
```

Out[3]:

```
5
```

### Updating the Value of an Existing Key–Value Pair

In [4]:

```
roman_numerals['X'] = 10
```

In [5]:

```
roman_numerals
```

Out[5]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10}
```

### Adding a New Key–Value Pair

In [6]:

```
roman_numerals['L'] = 50
```

In [7]:

```
roman_numerals
```

Out[7]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10, 'L': 50}
```

- String keys are case sensitive.
- Assigning to a nonexistent key inserts a new key–value pair.

## Removing a Key–Value Pair

In [8]:

```
del roman_numerals['III']
```

In [9]:

```
roman_numerals
```

Out[9]:

```
{'I': 1, 'II': 2, 'V': 5, 'X': 10, 'L': 50}
```

- Method **`pop`** returns the value for the removed key.

In [10]:

```
roman_numerals.pop('X')
```

Out[10]:

```
10
```

In [11]:

```
roman_numerals
```

Out[11]:

```
{'I': 1, 'II': 2, 'V': 5, 'L': 50}
```

## Attempting to Access a Nonexistent Key

In [12]:

```
roman_numerals['III']
```

```
---------------------------------------------------------------------
----
KeyError                                Traceback (most recent call l
ast)
<ipython-input-12-ccd50c7f0c8b> in <module>
----> 1 roman_numerals['III']

KeyError: 'III'
```

- Method **`get`** returns its argument’s corresponding value or `None` if the key is not found.
- IPython does not display anything for `None` .
- `get` with a second argument returns teh second argument if the key is not found.

In [13]:

```
roman_numerals.get('III')
```

In [14]:

```
roman_numerals.get('III', 'III not in dictionary')
```

Out[14]:

```
'III not in dictionary'
```

In [15]:

```
roman_numerals.get('V')
```

Out[15]:

```
5
```

## Testing Whether a Dictionary Contains a Specified Key

In [16]:

```
'V' in roman_numerals
```

Out[16]:

```
True
```

In [17]:

```
'III' in roman_numerals
```

Out[17]:

```
False
```

In [18]:

```
'III' not in roman_numerals
```

Out[18]:

```
True
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.4 Dictionary Methods `keys` and `values`

In [1]:

```
months = {'January': 1, 'February': 2, 'March': 3}
```

In [2]:

```
for month_name in months.keys():
    print(month_name, end='  ')
```

```
January  February  March
```

In [3]:

```
for month_number in months.values():
    print(month_number, end='  ')
```

```
1  2  3
```

### Dictionary Views

- Methods `items`, `keys` and `values` each return a **view** of a dictionary’s data.
- When you iterate over a **`view`**, it “sees” the dictionary’s **current contents**—it does **not** have its own copy of the data.

In [4]:

```
months_view = months.keys()
```

In [5]:

```
for key in months_view:
    print(key, end='  ')
```

```
January  February  March
```

In [6]:

```
months['December'] = 12
```

In [7]:

```
months
```

Out[7]:

```
{'January': 1, 'February': 2, 'March': 3, 'December': 12}
```

In [8]:

```
for key in months_view:
    print(key, end='  ')
```

```
January  February  March  December
```

## Converting Dictionary Keys, Values and Key–Value Pairs to Lists

In [9]:

```
list(months.keys())
```

Out[9]:

```
['January', 'February', 'March', 'December']
```

In [10]:

```
list(months.values())
```

Out[10]:

```
[1, 2, 3, 12]
```

In [11]:

```
list(months.items())
```

Out[11]:

```
[('January', 1), ('February', 2), ('March', 3), ('December', 12)]
```

## Processing Keys in Sorted Order

In [12]:

```
for month_name in sorted(months.keys()):
     print(month_name, end='  ')
```

```
December  February  January  March
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.5 Dictionary Comparisons

- `==` is `True` if both dictionaries have the same key–value pairs, ***regardless* of the order in which those key–value pairs were added to each dictionary.**

In [1]:

```
country_capitals1 = {'Belgium': 'Brussels',
                     'Haiti': 'Port-au-Prince'}
```

In [2]:

```
country_capitals2 = {'Nepal': 'Kathmandu',
                     'Uruguay': 'Montevideo'}
```

In [3]:

```
country_capitals3 = {'Haiti': 'Port-au-Prince',
                     'Belgium': 'Brussels'}
```

In [4]:

```
country_capitals1 == country_capitals2
```

Out[4]:

```
False
```

In [5]:

```
country_capitals1 == country_capitals3
```

Out[5]:

```
True
```

In [6]:

```
country_capitals1 != country_capitals2
```

Out[6]:

```
True
```

This content is based on Chapter 6 of the book **Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud** (https://amzn.to/2VvdnxE).

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.6 Example: Dictionary of Student Grades

- Script with a dictionary that represents an instructor’s grade book.
- Maps each student’s name (a string) to a list of integers containing that student’s grades on three exams.

```
# fig06_01.py
"""Using a dictionary to represent an instructor's grade book."""
grade_book = {
    'Susan': [92, 85, 100],
    'Eduardo': [83, 95, 79],
    'Azizi': [91, 89, 82],
    'Pantipa': [97, 91, 92]
}

all_grades_total = 0
all_grades_count = 0

for name, grades in grade_book.items():
    total = sum(grades)
    print(f'Average for {name} is {total/len(grades):.2f}')
    all_grades_total += total
    all_grades_count += len(grades)

print(f"Class's average is: {all_grades_total / all_grades_count:.2f}")
```

```
Average for Susan is 92.33
Average for Eduardo is 85.67
Average for Azizi is 87.33
Average for Pantipa is 93.33
Class's average is: 89.67
```

In [66]:

```
run fig06_01.py
```

```
Average for Susan is 92.33
Average for Eduardo is 85.67
Average for Azizi is 87.33
Average for Pantipa is 93.33
Class's average is: 89.67
```

This content is based on Chapter 6 of the book **Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud** (https://amzn.to/2VvdnxE).

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.7 Example: Word Counts

- Script that builds a dictionary to count the number of occurrences of each word in a **tokenized** string.
- Python automatically concatenates strings separated by whitespace in parentheses.

```python
# fig06_02.py
"""Tokenizing a string and counting unique words."""

text = ('this is sample text with several words '
        'this is more sample text with some different words')

word_counts = {}

# count occurrences of each unique word
for word in text.split():
    if word in word_counts:
        word_counts[word] += 1  # update existing key-value pair
    else:
        word_counts[word] = 1  # insert new key-value pair

print(f'{"WORD":<12}COUNT')

for word, count in sorted(word_counts.items()):
    print(f'{word:<12}{count}')

print('\nNumber of unique words:', len(word_counts))
```

In [1]:

```
run fig06_02.py
```

```
WORD        COUNT
different   1
is          2
more        1
sample      2
several     1
some        1
text        2
this        2
with        2
words       2

Number of unique words: 10
```

## Python Standard Library Module `collections`

- The Python Standard Library already contains the counting functionality shown above.
- A **`Counter`** is a customized dictionary that receives an iterable and summarizes its elements.

In [2]:

```
from collections import Counter
```

In [3]:

```
text = ('this is sample text with several words '
        'this is more sample text with some different words')
```

In [4]:

```
counter = Counter(text.split())
```

In [5]:

```
for word, count in sorted(counter.items()):
    print(f'{word:<12}{count}')
```

```
different   1
is          2
more        1
sample      2
several     1
some        1
text        2
this        2
with        2
words       2
```

In [6]:

```
print('Number of unique keys:', len(counter.keys()))
```

```
Number of unique keys: 10
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.8 Dictionary Method `update`

- Can insert and update key–value pairs.
- Method `update` also can receive an iterable object containing key–value pairs, such as a list of two-element tuples.

In [1]:

```
country_codes = {}
```

In [2]:

```
country_codes.update({'South Africa': 'za'})
```

In [3]:

```
country_codes
```

Out[3]:

```
{'South Africa': 'za'}
```

- Purposely inserting an incorrect value:

In [4]:

```
country_codes.update(Australia='ar')
```

In [5]:

```
country_codes
```

Out[5]:

```
{'South Africa': 'za', 'Australia': 'ar'}
```

- Correcting the incorrect value:

In [6]:

```
country_codes.update(Australia='au')
```

In [7]:

```
country_codes
```

Out[7]:

```
{'South Africa': 'za', 'Australia': 'au'}
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.2.9 Dictionary Comprehensions

- Convenient notation for quickly generating dictionaries, often by **mapping** one dictionary to another.
- The expression to the left of the `for` clause specifies a **key–value pair of the form *key* `:` *value***.
- In a dictionary with ***unique* values**, you can **reverse** the key–value pair mappings.

In [1]:

```
months = {'January': 1, 'February': 2, 'March': 3}
```

In [2]:

```
months2 = {number: name for name, number in months.items()}
```

In [3]:

```
months2
```

Out[3]:

```
{1: 'January', 2: 'February', 3: 'March'}
```

- A dictionary comprehension also can map a dictionary's values to new values.

In [4]:

```
grades = {'Sue': [98, 87, 94], 'Bob': [84, 95, 91]}
```

In [5]:

```
grades2 = {k: sum(v) / len(v) for k, v in grades.items()}
```

In [6]:

```
grades2
```

Out[6]:

```
{'Sue': 93.0, 'Bob': 90.0}
```

In [7]:

```
##########################################################################
# (C) Copyright 2019 by Deitel & Associates, Inc. and                    #
# Pearson Education, Inc. All Rights Reserved.                           #
#                                                                        #
# DISCLAIMER: The authors and publisher of this book have used their     #
# best efforts in preparing the book. These efforts include the          #
# development, research, and testing of the theories and programs        #
# to determine their effectiveness. The authors and publisher make       #
# no warranty of any kind, expressed or implied, with regard to these    #
# programs or to the documentation contained in these books. The authors #
# and publisher shall not be liable in any event for incidental or       #
# consequential damages in connection with, or arising out of, the       #
# furnishing, performance, or use of these programs.                     #
##########################################################################
```

# 6.3 Sets

- A set is an unordered collection of **unique values**.
- May contain **only immutable objects**, like strings, `int` s, `float` s and tuples that contain only immutable elements.
- Sets do not support indexing and slicing.

## Creating a Set with Curly Braces

- Duplicates are ignored, making sets great for **duplicate elimination**.

In [1]:

```
colors = {'red', 'orange', 'yellow', 'green', 'red', 'blue'}
```

- Though the output below is sorted, sets are **unordered**—do not write order-dependent code.

In [2]:

```
colors
```

Out[2]:

```
{'blue', 'green', 'orange', 'red', 'yellow'}
```

## Determining a Set’s Length

In [3]:

```
len(colors)
```

Out[3]:

```
5
```

## Checking Whether a Value Is in a Set

In [4]:

```
'red' in colors
```

Out[4]:

```
True
```

In [5]:

```
'purple' in colors
```

Out[5]:

```
False
```

In [6]:

```
'purple' not in colors
```

Out[6]:

```
True
```

## Iterating Through a Set

- There’s no significance to the iteration order.

In [7]:

```
for color in colors:
    print(color.upper(), end=' ')
```

```
YELLOW BLUE GREEN RED ORANGE
```

## Creating a Set with the Built-In `set` Function

In [8]:

```
numbers = list(range(10)) + list(range(5))
```

In [9]:

```
numbers
```

Out[9]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
```

In [10]:

```
set(numbers)
```

Out[10]:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- To create an empty set, must use the **`set()`** , because **`{}` represents an empty dictionary**.
- Python displays an empty set as `set()` to avoid confusion with an empty dictionary ( `{}` ).

In [11]:

```
set()
```

Out[11]:

```
set()
```

## Frozenset: An Immutable Set Type

- **Sets are *mutable***.
- **Set *elements* must be *immutable***; therefore, a set cannot have other sets as elements.
- A **frozenset** is an *immutable* set—it cannot be modified after you create it, so a set *can* contain frozensets as elements.
- The built-in function **`frozenset`** creates a frozenset from any iterable.

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.3.1 Comparing Sets

```
{1, 3, 5} == {3, 5, 1}
```

```
True
```

```
{1, 3, 5} != {3, 5, 1}
```

```
False
```

- `<` tests whether the set to its left is a **proper subset** of the one to its right—all the elements in the left operand are in the right operand, and **the sets are not equal**.

```
{1, 3, 5} < {3, 5, 1}
```

```
False
```

```
{1, 3, 5} < {7, 3, 5, 1}
```

```
True
```

- The `<=` operator tests whether the set to its left is an **improper subset** of the one to its right—that is, all the elements in the left operand are in the right operand, and **the sets might be equal**:

```
{1, 3, 5} <= {3, 5, 1}
```

```
True
```

```
{1, 3} <= {3, 5, 1}
```

```
True
```

- You may also check for an improper subset with the set method **`issubset`**:

```
{1, 3, 5}.issubset({3, 5, 1})
```

```
True
```

```
{1, 2}.issubset({3, 5, 1})
```

```
False
```

- Similarly, you may also check for a **proper superset** with `>` and **improper supersets** with `>=` or set method **`issuperset`**.

```
{1, 3, 5} > {3, 5, 1}

     False

{1, 3, 5, 7} > {3, 5, 1}

     True

{1, 3, 5} >= {3, 5, 1}

     True

{1, 3, 5} >= {3, 1}

     True

{1, 3} >= {3, 1, 7}

     False

{1, 3, 5}.issuperset({3, 5, 1})

     True

{1, 3, 5}.issuperset({3, 2})

     False
```

- Argument to `issubset` or `issuperset` can be *any* iterable.
- For a non-set iterable argument, the methods first convert the iterable to a set, then perform the operation.

---

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Prodotti Colab a pagamento - Annulla i contratti qui

✓ 0 s data/ora di completamento: 12:39 ● ×

## 6.3.2 Mathematical Set Operations

### Union

- The **union** of two sets is a set consisting of all the unique elements from both sets.
- The union operator requires two sets, but method `union` may receive any iterable as its argument (this is true for subsequent methods in this section as well).

In [1]:

```
{1, 3, 5} | {2, 3, 4}
```

Out[1]:

```
{1, 2, 3, 4, 5}
```

In [2]:

```
{1, 3, 5}.union([20, 20, 3, 40, 40])
```

Out[2]:

```
{1, 3, 5, 20, 40}
```

### Intersection

The **intersection** of two sets is a set consisting of all the unique elements that the two sets have in common.

In [3]:

```
{1, 3, 5} & {2, 3, 4}
```

Out[3]:

```
{3}
```

In [4]:

```
{1, 3, 5}.intersection([1, 2, 2, 3, 3, 4, 4])
```

Out[4]:

```
{1, 3}
```

### Difference

The **difference** between two sets is a set consisting of the elements in the left operand that are not in the right operand.

In [5]:

```
{1, 3, 5} - {2, 3, 4}
```

Out[5]:

```
{1, 5}
```

In [6]:

```
{1, 3, 5, 7}.difference([2, 2, 3, 3, 4, 4])
```

Out[6]:

```
{1, 5, 7}
```

## Symmetric Difference

The **symmetric difference** between two sets is a set consisting of the elements of both sets that are not in common with one another.

In [7]:

```
{1, 3, 5} ^ {2, 3, 4}
```

Out[7]:

```
{1, 2, 4, 5}
```

In [8]:

```
{1, 3, 5, 7}.symmetric_difference([2, 2, 3, 3, 4, 4])
```

Out[8]:

```
{1, 2, 4, 5, 7}
```

## Disjoint

Two sets are **disjoint** if they do not have any common elements.

In [9]:

```
{1, 3, 5}.isdisjoint({2, 4, 6})
```

Out[9]:

```
True
```

In [10]:

```
{1, 3, 5}.isdisjoint({4, 6, 1})
```

Out[10]:

```
False
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.3.3 Mutable Set Operators and Methods

### Mutable Mathematical Set Operations

In [1]:

```
numbers = {1, 3, 5}
```

In [2]:

```
numbers |= {2, 3, 4}
```

In [3]:

```
numbers
```

Out[3]:

```
{1, 2, 3, 4, 5}
```

In [4]:

```
numbers.update(range(10))
```

In [5]:

```
numbers
```

Out[5]:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Other mutable set methods:

- intersection augmented assignment `&=`
- difference augmented assignment `-=`
- symmetric difference augmented assignment `^=`

Corresponding methods with iterable arguments:

- `intersection_update`
- `difference_update`
- `symmetric_difference_update`

### Methods for Adding and Removing Elements

- Set method **`add`** inserts its argument if the argument is *not* already in the set; otherwise, the set remains unchanged.

In [6]:

```
numbers.add(17)
```

In [7]:

```
numbers.add(3)
```

In [8]:

```
numbers
```

Out[8]:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 17}
```

- Method **`remove`** removes its argument from the set—raises a **`KeyError`** if the value is not in the set.

In [9]:

```
numbers.remove(3)
```

In [10]:

```
numbers
```

Out[10]:

```
{0, 1, 2, 4, 5, 6, 7, 8, 9, 17}
```

- Method **`discard`** also removes its argument from the set but **does not cause an exception if the value is not in the set.**
- Can remove an *arbitrary* set element and return it with **`pop`** .

In [11]:

```
numbers.pop()
```

Out[11]:

```
0
```

In [12]:

```
numbers
```

Out[12]:

```
{1, 2, 4, 5, 6, 7, 8, 9, 17}
```

In [13]:

```
numbers.clear()  # empty the set
```

In [14]:

```
numbers
```

Out[14]:

```
set()
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

## 6.3.4 Set Comprehensions

- Define set comprehensions in curly braces.

In [1]:

```
numbers = [1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10, 10]
```

In [2]:

```
evens = {item for item in numbers if item % 2 == 0}
```

In [3]:

```
evens
```

Out[3]:

```
{2, 4, 6, 8, 10}
```

DISCLAIMER: The authors and publisher of this book have used their best efforts in preparing the book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.