CHAPTER 9

Dictionaries and Sets

Topics

- **Dictionaries**
- **Sets**
- **Serializing Objects**

Dictionaries

- **<u>Dictionary</u>: object that stores a collection of data**
  - Each element consists of a *key* and a *value*
    - Often referred to as *mapping* of key to value
    - Key must be unique – like an Employee ID or Name etc.
    - Key must be an immutable object
  - To retrieve a specific value, use the key associated with it
- This is similar to the process of looking up a word in the dictionary, where the <u>words are keys</u> and the <u>definitions are values</u>.
  - Format for creating a dictionary

    dictionary ={*key1*:*val1*, *key2*:*val2*}

<u>Example:</u>

```
phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
```

Creating a dictionary named phonebook with three elements. Here keys can be considered as names and values are corresponding phone numbers each names.

Retrieving a Value from a Dictionary

- **Elements in dictionary are unsorted**

```
>>> phonebook
{'Chris': '555-1111', 'Katie': '555-2222', 'Joanne': '555-3333'}
```

- **General format for retrieving a value from dictionary:**
  dictionary_name[*key*]

```
>>> phonebook['Chris']
'555-1111'
```

  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised

```
>>> phonebook['Kathryn']
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    phonebook['Kathryn']
KeyError: 'Kathryn'
```

- **Test whether a key is in a dictionary using the `in` and `not in` operators**
  - Helps prevent `KeyError` exceptions

Example: Using in Operator for a Dictionary

- **Create a dictionary named** `age_dict` **with some where keys are the names and ages are the values. Write a program that user search a person from this data.**

```
age_dict={'Chris':27,'Katie':35,'Joanne':28,'Tom':22}
# Input the name from the user
name=input('Enter a name to be searched:')

# First check if it is in the dictonary
# Then retrive the data if it is in the dictionary
if name in age_dict:
    print(name,'is',age_dict[name],'years old.')
else:
    print(name,'can not be found!')
```

Program Output

```
Enter a name to be searched:Katie
Katie is 35 years old.
```

Program Output

```
Enter a name to be searched:John
John can not be found!
```

Adding Elements to an Existing Dictionary

- **Dictionaries are mutable objects**
- **To add a new key-value pair:**

dictionary[key] = value

```
>>> ages = {'Chris':23,'Katie':37,'Andrew':16}
>>> ages['John']=18
>>> ages
{'Chris': 23, 'Katie': 37, 'Andrew': 16, 'John': 18}
```

- **This can be also used to modify an existing element**
  - If key exists in the dictionary, the value associated with it will be changed

```
>>> ages['Katie']=38
>>> ages['Chris']=ages['Chris']+1
>>> ages['Andrew']+=3
>>> ages
{'Chris': 24, 'Katie': 38, 'Andrew': 19, 'John': 18}
```

Deleting Elements From a Dictionary

- **To delete a key-value pair:**

  `del dictionary[key]`

  - If key is not in the dictionary, `KeyError` exception is raised

```
>>> ages
{'Chris': 24, 'Katie': 38, 'Andrew': 19, 'John': 18}
>>> del ages['Andrew']
>>> ages
{'Chris': 24, 'Katie': 38, 'John': 18}
>>> del ages['Tom']
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    del ages['Tom']
KeyError: 'Tom'
```

- **Test whether a key is in a dictionary using the `in` and `not in` operators**
  - Helps prevent `KeyError` exceptions

Getting the Number of Elements

- **`len` function: used to obtain the number of elements in a dictionary**

```
>>> ages = {'Chris':23,'Katie':37,'Andrew':16}
>>> print(len(ages))
3
>>> ages['John']=15
>>> print(len(ages))
4
>>> ages['Amy']=28
>>> print(len(ages))
5
>>> del ages['Chris']
>>> print(len(ages))
4
>>> ages
{'Katie': 37, 'Andrew': 16, 'John': 15, 'Amy': 28}
```

Several Data in a Dictionary Value

- **Keys must be immutable and unique objects, but associated values can be any type of object**
  - One dictionary can include keys of several different immutable types

```
>>> test_scores = { 'Kayla' : [88, 92, 100],'Luis' : [95, 74, 81],
                   'Sophie' : [72, 88, 91], 'Ethan' : [70, 75, 78] }
>>> print(test_scores['Kayla'])
[88, 92, 100]
>>> test_scores['Kayla']=[55,95,96]
>>> print(test_scores)
{'Kayla': [55, 95, 96], 'Luis': [95, 74, 81], 'Sophie': [72, 88, 91], 'Ethan': [70, 75, 78]}
>>> test_scores['Tom']=[100,100,100]
>>> print(test_scores)
{'Kayla': [55, 95, 96], 'Luis': [95, 74, 81], 'Sophie': [72, 88, 91], 'Ethan': [70, 75, 78], 'Tom'
: [100, 100, 100]}
>>>
>>> test_scores['Tom'][0]=95
>>> print(test_scores['Tom'])
[95, 100, 100]
>>>
```

Mixing Data Types in a Dictionary

- **Values stored in a single dictionary can be of different types**

```
>>> employee = {'name':'Kevin Smith', 'id':12345, 'payrate':25.75}
>>> print(employee)
{'name': 'Kevin Smith', 'id': 12345, 'payrate': 25.75}
>>> print(employee['name'])
Kevin Smith
>>> print(employee['payrate'])
25.75
>>> employee['payrate']=float(input('Enter new Payrate:'))
Enter new Payrate:28.35
>>> print(employee)
{'name': 'Kevin Smith', 'id': 12345, 'payrate': 28.35}
>>>
```

Creating an Empty Dictionary

- **To create an empty dictionary:**
  - Use `{}`
  - Elements can be added to the dictionary as program executes

```
>>> phonebook = {}
>>> print(phonebook)
{}
>>>
>>> phonebook['Adam']='555-4536'
>>>
>>> name=input('Enter a name:')
Enter a name:John
>>> phone=input('Enter phone number:')
Enter phone number:555-1245
>>> phonebook[name]=phone
>>>
>>> print(phonebook)
{'Adam': '555-4536', 'John': '555-1245'}
```

- Also may use built-in function `dict()`

  `phonebook = dict()` same as `phonebook = {}`

Using `for` Loop to Iterate Over a Dict.

- **Use a `for` loop to iterate over a dictionary**
  - `for` repetition iterates through each key values in the order
  - **General format:** `for` *`key`* `in` *`dictionary`*`:`

Example:

```
phonebook = {'Chris':'555-1111',
             'Katie':'555-2222',
             'Joanne':'555-3333'}

# Let's print the keys by iterating through phonebook
for key in phonebook:
    print(key)
```

```
Chris
Katie
Joanne
```

```
# Let's print the values again by using for loop
for key in phonebook:
    print(phonebook[key])
```

```
555-1111
555-2222
555-3333
```

Some Dictionary Methods

Dictionary objects have several methods. In this section, we look at some of the most useful ones, which are summarized in Table 9–1.

Table 9-1 Some of the dictionary methods

Page 469

Method	Description
`clear`	Clears the contents of a dictionary.
`get`	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
`items`	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
`keys`	Returns all the keys in a dictionary as a sequence of tuples.
`pop`	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
`popitem`	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.
`values`	Returns all the values in the dictionary as a sequence of tuples.

`clear` Dictionary Method

- **`clear` method: deletes all the elements in a dictionary, leaving it empty**
  - Format: *`dictionary`*`.clear()`

```
>>> test_scores={'Kyle': [40, 92], 'Tom': [72, 91], 'Joe': [70, 75]}
>>> print(test_scores)
{'Kyle': [40, 92], 'Tom': [72, 91], 'Joe': [70, 75]}
>>> test_scores.clear()
>>> print(test_scores)
{}
>>>
```

`get` Dictionary Method

- **<u>`get` method</u>: gets a value associated with specified key from the dictionary**
  - Format: *`dictionary`*`.get(`*`key, default`*`)`
    - *`default`* is returned if *`key`* is not found

```
>>> test_scores={'Kyle': [40, 92], 'Tom': [72, 91], 'Joe': [70, 75]}
>>> test_scores.get('Kyle')
[40, 92]
>>> test_scores.get('Tom','Key not Found')
[72, 91]
>>> test_scores.get('John','Key not Found')
'Key not Found'
```

  - Any text/message can be used as second parameter just in case the key not found.
  - Alternative to `[]` operator – THIS MAY RAISE EXCEPTION
    - Cannot raise `KeyError` exception

`items` Dictionary Method

- **`items` method: returns all the dictionaries keys and associated values**
  - Format: *`dictionary`*`.items()`
  - Returned as a *dictionary view*
    - Each element in dictionary view is a tuple which contains a key and its associated value
    - Use a `for` loop to iterate over the tuples in the sequence
      - Can use a variable which receives a tuple, or can use two variables which receive key and value

```
>>> phonebook = {'Chris':'555-1111','Kaie':'555-2222','Joe':'555-3333'}
>>> for key, value in phonebook.items():
        print(key, value)


Chris 555-1111
Kaie 555-2222
Joe 555-3333
```

`keys` Dictionary Method

- **`keys` method: returns all the dictionaries keys as a sequence**
  - General Format: *`dictionary`*`.keys()`

```
>>> phonebook = {'Chris':'555-1111','Kaie':'555-2222','Joe':'555-3333'}
>>> phonebook.keys()
dict_keys(['Chris', 'Kaie', 'Joe'])
```

  - Can use iteration through these key values.

```
>>> phonebook = {'Chris':'555-1111','Kaie':'555-2222','Joe':'555-3333'}
>>> for key in phonebook.keys():
        print(key)
```

```
Chris
Kaie
Joe
```

Note: by using the obtained keys `phonebook[key]` or `phonebook.get(key)` can be used to get the values for each key.

`values` Dictionary Method

- **`values` method: returns all the dictionaries values as a sequence**
  - Format: *`dictionary`*`.values()`

```
>>> phonebook = {'Chris':'555-1111',
        'Katie':'555-2222','Joanne':'555-3333'}
>>> phonebook.values()
dict_values(['555-1111', '555-2222', '555-3333'])
```

  - Use a `for` loop to iterate over the values

```
>>> phonebook = {'Chris':'555-1111',
        'Katie':'555-2222','Joanne':'555-3333'}
>>> for value in phonebook.values():
        print(value)


555-1111
555-2222
555-3333
```

pop Dictionary Method

- **<u>pop method</u>: returns value associated with specified key and <u>removes that key-value pair</u> from the dictionary**
  - General Format: *dictionary*.pop(*key*, *default*)
    - *default* is returned if *key* is not found

```
>>> ages = {'Chris':23,'Katie':35,'Joe':42}
>>> age = ages.pop('Katie','Not Found!')
>>> age
35
>>> ages
{'Chris': 23, 'Joe': 42}
>>> age = ages.pop('Tom','Not Found!')
>>> age
'Not Found!'
>>>
```

`popitem` Dictionary Method

- **`popitem` method: returns a randomly selected key-value pair and removes that key-value pair from the dictionary**
  - General Format: *`dictionary`*`.popitem()`
  - Key-value pair returned as a tuple – returned value can be assigned to key and value multiple value

```
>>> phonebook = {'Chris':'555-1111',
        'Katie':'555-2222','Joanne':'555-3333'}
>>> k , v = phonebook.popitem()
>>> print( k , v )
Joanne 555-3333
>>>
>>> phonebook
{'Chris': '555-1111', 'Katie': '555-2222'}
```

- The `popitem` method raises a `KeyError` exception if it is called on an empty dictionary
- *Changed in version 3.7: In prior versions, popitem() would return an arbitrary key/value pair. Now it pops the last item in the dictionary.*

In the Spotlight:
Storing Names and Birthdays in a Dictionary

Consider a program that keeps your friends' names and birthdays in a dictionary. Each entry in the dictionary uses a friend's name as the key, and that friend's birthday as the value.

- The program displays a menu that allows the user to make one of the following choices:

```
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program
```

The program initially starts with an empty dictionary, so you have to choose item 2 from the menu to add a new entry. Once you have added a few entries, you can start using other options in your program.

Program Organization:

Main Program will control the whole program. Following functions will included in the program.

`get_menu_choice` function: displays menu and gets the choice and send to the main.

`look_up` function: gets person name from user and displays the birthday.

`add` function: gets person info to be added from user and adds it to `birthdays` dictionary.

`change` function: gets person name to be modified from user and modifies it in the dictionary.

`delete` function: gets person name to be deleted from user and deletes it from the dictionary.

Not: Main will create the birthdays dictionary and send it to functions to be used.

Birthdays Program: `main` function

`main` function:

```
# This program uses a dictionary to keep friends'
# names and birthdays.

# main function
def main():
    # Create an empty dictionary.
    birthdays = {}

    # Initialize a variable for the user's choice.
    choice = 0

    while choice != 5:
        # Get the user's menu choice.
        choice = get_menu_choice()

        # Process the choice.
        if choice == 1:
            look_up(birthdays)
        elif choice == 2:
            add(birthdays)
        elif choice == 3:
            change(birthdays)
        elif choice == 4:
            delete(birthdays)

# Call the main function.
main()
```

Birthdays Program: `get_menu_choice`

`get_menu_choice` function:

```
# The get_menu_choice function displays the menu
# and gets a validated choice from the user.
def get_menu_choice():
    print()
    print('Friends and Their Birthdays')
    print('---------------------------')
    print('1. Look up a birthday')
    print('2. Add a new birthday')
    print('3. Change a birthday')
    print('4. Delete a birthday')
    print('5. Quit the program')
    print()

    # Get the user's choice.
    choice = int(input('Enter your choice: '))

    # Validate the choice.
    while choice < 1 or choice > 5:
        choice = int(input('Enter a valid choice: '))

    # return the user's choice.
    return choice
```

Birthdays Prog: `look_up` – `add` functions

`look_up` function:

```
# The look_up function looks up a name in the
# birthdays dictionary.
def look_up(birthdays):
    # Get a name to look up.
    name = input('Enter a name: ')

    # Look it up in the dictionary.
    print(birthdays.get(name, 'Not found.'))
```

`add` function:

```
# The add function adds a new entry into the
# birthdays dictionary.
def add(birthdays):
    # Get a name and birthday.
    name = input('Enter a name: ')
    bday = input('Enter a birthday: ')

    # If the name does not exist, add it.
    if name not in birthdays:
        birthdays[name] = bday
    else:
        print('That entry already exists.')
```

Birthdays Prog: `change-delete` functions

`change` function:

```
# The change function changes an existing
# entry in the birthdays dictionary.
def change(birthdays):
    # Get a name to look up.
    name = input('Enter a name: ')

    if name in birthdays:
        # Get a new birthday.
        bday = input('Enter the new birthday: ')

        # Update the entry.
        birthdays[name] = bday
    else:
        print('That name is not found.')
```

`delete` function:

```
# The delete function deletes an entry from the
# birthdays dictionary.
def delete(birthdays):
    # Get a name to look up.
    name = input('Enter a name: ')

    # If the name is found, delete the entry.
    if name in birthdays:
        del birthdays[name]
    else:
        print('That name is not found.')
```

Birthdays Program Run

```
Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 2
Enter a name: Sam Vyne
Enter a birthday: 01/01/2000

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 2
Enter a name: Ahmet Can
Enter a birthday: 06/26/1995

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 2
Enter a name: John Taylor
Enter a birthday: 07/13/2003

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1
Enter a name: Sam Vyne
01/01/2000
```

Four Data are added to dictionary.

```
Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1
Enter a name: Sam Vyne
01/01/2000

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 3
Enter a name: Sam Vyne
Enter the new birthday: 03/03/2003

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1
Enter a name: Sam Vyne
03/03/2003

Friends and Their Birthday
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 5
>>>
```

9.1 An element in a dictionary has two parts. What are they called?

key and value

9.2 Which part of a dictionary element must be immutable?

key part of a dictionary is immutable

9.3 Suppose 'start' : 1472 is an element in a dictionary. What is the key? What is the value?

key is 'start' and value is 1472

9.4 Suppose a dictionary named employee has been created. What does the following statement do?

```
employee['id'] = 54321
```

It modifies the id as 54321 or adds it to the employee dictionary if it is not already exist.

9.5 What will the following code display?

```
stuff = {1 : 'aaa', 2 : 'bbb', 3 : 'ccc'}
print(stuff[3])
```

It displays `ccc`

9.6 How can you determine whether a key-value pair exists in a dictionary?

by using `in` or `not in` operators.

9.7 Suppose a dictionary named inventory exists. What does the following statement do?

```
del inventory[654]
```

It deletes/removes the element with key 654 from the inventory dictionary.

9.8 What will the following code display?

```
stuff = {1 : 'aaa', 2 : 'bbb', 3 : 'ccc'}
print(len(stuff))
```

It displays 3 which is the number of elements in the stuff dictionary.

9.9 What will the following code display?

```
stuff = {1 : 'aaa', 2 : 'bbb', 3 : 'ccc'}
for k in stuff:
        print(k)
```

It displays the key values in the dictionary in separate lines.

9.10 What is the difference between the dictionary methods pop and popitem?

pop returns removes an element with a given key. popitem also does the same operation but it picks the element randomly.

9.11 What does the items method return?

Returns all the dictionaries keys and associated values as a tuple

9.12 What does the keys method return?

Returns all the dictionary keys as a sequence.

9.13 What does the values method return?

Returns all the dictionary values as a sequence.

Algorithm Workbench

1. Write a statement that creates a dictionary containing the following key-value pairs:

   'a' : 1
   'b' : 2
   'c' : 3

```
my_dict = {'a':1,'b':2,'c',3}
```

3. Assume the variable `dct` references a dictionary. Write an `if` statement that determines whether the key `'James'` exists in the dictionary. If so, display the value that is associated with that key. If the key is not in the dictionary, display a message indicating so.

```
if 'James' in dct:
        print(dct['James'])
else:
        print('James not found!)
```

4. Assume the variable `dct` references a dictionary. Write an `if` statement that determines whether the key `'Jon'` exists in the dictionary. If so, assign the value of `'Jon'` to a key of `'John'`, and then delete `'Jon'` and its associated value.

```
if 'Jon' in dct:
        dct['John'] = dct['Jon']
        del dtc['Jon']
```

Programming Exercises

1. Galilean Moons of Jupiter

Write a program that creates a dictionary containing the names of the Galilean moons of Jupiter as keys and their mean radiuses (in kilometers) as values. The dictionary should contain the following key-value pairs:

Moon Name (key)	Mean Radius (value)
Io	1821.6
Europa	1560.8
Ganymede	2634.1
Callisto	2410.3

The program should also create a dictionary containing the moon names and their surface gravities (in meters per second squared). The dictionary should contain the following key-value pairs:

Moon Name (key)	Surface Gravity (value)
Io	1.796
Europa	1.314
Ganymede	1.428
Callisto	1.235

The program should also create a dictionary containing the moon names and their orbital periods (in days). The dictionary should contain the following key-value pairs:

Moon Name (key)	Orbital Period (value)
Io	1.769
Europa	3.551
Ganymede	7.154
Callisto	16.689

The program should let the user enter the name of a Galilean moon of Jupiter, then it should display the moon's mean radius, surface gravity and orbital period.

Programming Exercises

1. Galilean Moons of Jupiter

```
# The main function.
def main():
    # Initialize dictionaries.
    mean_radius = {'io':1821.6, 'europa':1560.8, \
                   'ganymede':2634.1, 'callisto':2410.3}

    surface_gravity = {'io':1.796, 'europa':1.314, \
                       'ganymede':1.428, 'callisto':1.235}


    orbital_period = {'io':1.769, 'europa':3.551, \
                      'ganymede':7.154, 'callisto':16.689}


    # Get input from user.
    moon = input('Enter name of Galilean moon of Jupiter: ')

    # Convert to lowercase for reliable matching in dictionaries.
    moon = moon.lower()

    # Show error message if name does not exist.
    # Otherwise, display details of specified moon.
    if moon not in mean_radius:
        print(moon.title(), 'is an invalid moon name.')
    else:
        print('Details of', moon.title(), 'are:')
        print('Mean Radius:', mean_radius[moon], 'km')
        print('Surface Gravity:', surface_gravity[moon], 'm/s²')
        print('Orbital Period:', orbital_period[moon], 'days')

# Call the main function.
main()
```

Program Output

```
Enter name of Galilean moon of Jupiter: europa
Details of Europa are:
Mean Radius: 1560.8 km
Surface Gravity: 1.314 m/s²
Orbital Period: 3.551 days
```

Program Output

```
Enter name of Galilean moon of Jupiter: Europe
Europe is an invalid moon name.
```

Programming Exercises

2. Capital Quiz

Write a program that creates a dictionary containing the U.S. states as keys, and their capitals as values. (Use the Internet to get a list of the states and their capitals.) The program should then randomly quiz the user by displaying the name of a state and asking the user to enter that state's capital. The program should keep a count of the number of correct and incorrect responses. (As an alternative to the U.S. states, the program can use the names of countries and their capitals.)

```
# US States and Capitals Quiz Program

def main():
    # Initialize dictionary
    capitals = {'Alabama':'Montgomery', 'Alaska':'Juneau',
                'Arizona':'Phoenix', 'Arkansas':'Little Rock',
                'California':'Sacramento', 'Colorado':'Denver',
                'Connecticut':'Hartford', 'Delaware':'Dover',
                'Florida':'Tallahassee', 'Georgia':'Atlanta',
                'Hawaii':'Honolulu', 'Idaho':'Boise',
                'Illinois':'Springfield', 'Indiana':'Indianapolis',
                'Iowa':'Des Moines', 'Kansas':'Topeka',
                'Kentucky':'Frankfort', 'Louisiana':'Baton Rouge',
                'Maine':'Augusta', 'Maryland':'Annapolis',
                'Massachusetts':'Boston', 'Michigan':'Lansing',
                'Minnesota':'Saint Paul', 'Mississippi':'Jackson',
                'Missouri':'Jefferson City', 'Montana':'Helena',
                'Nebraska':'Lincoln', 'Nevada':'Carson City',
                'New Hampshire':'Concord', 'New Jersey':'Trenton',
                'New Mexico':'Santa Fe', 'New York':'Albany',
                'North Carolina':'Raleigh', 'North Dakota':'Bismarck',
                'Ohio':'Columbus', 'Oklahoma':'Oklahoma City',
                'Oregon':'Salem', 'Pennsylvania':'Harrisburg',
                'Rhode Island':'Providence', 'South Carolina':'Columbia',
                'South Dakota':'Pierre', 'Tennessee':'Nashville',
                'Texas':'Austin', 'Utah':'Salt Lake City',
                'Vermont':'Montpelier', 'Virginia':'Richmond',
                'Washington':'Olympia', 'West Virginia':'Charleston',
                'Wisconsin':'Madison', 'Wyoming':'Cheyenne'}
```

Programming Exercises

2. Capital Quiz

```
# US States and Capitals Quiz Program

def main():
    # Initialize dictionary
    capitals = {'Alabama':'Montgomery', 'Alaska':'Juneau',....}

    # Local variables
    correct = 0
    wrong = 0
    next_question = True

    # Continue until user quits the game.
    while next_question:

        # Randomly Select the names of the states
        state , capital = capitals.popitem()

        # Get user solution.
        user_solution = input('What is the capital of ' + \
                              state + \
                              '? (or enter 0 to quit): ')

        # User wants to quit the game.
        if user_solution == '0':
            next_question = False
            print('You had', correct, 'correct responses and',\
                  wrong, 'incorrect responses.')
        # User solution is correct.
        elif user_solution == capital:
            correct = correct + 1
            print('That is correct.')
        # User solution is incorrect.
        else:
            wrong = wrong + 1
            print('That is incorrect.')

# Call the main function.
main()
```

Program Output

```
What is the capital of Wyoming? (or enter 0 to quit): Madison
That is incorrect.
What is the capital of Wisconsin? (or enter 0 to quit): Madison
That is correct.
What is the capital of West Virginia? (or enter 0 to quit): Charlestor
That is correct.
What is the capital of Washington? (or enter 0 to quit): Olympia
That is correct.
What is the capital of Virginia? (or enter 0 to quit): 0
You had 3 correct responses and 1 incorrect responses.
```

Programming Exercises

5. Random Number Frequencies

Write a program that generates 100 random numbers between 1 and 10. The program should store the frequency of each number generated in a dictionary with the number as the key and the amount of times it has occurred as the value. For example, if the program generates the number 6 a total of 11 times, the dictionary will contain a key of 6 with an associated value of 11. Once all of the numbers have been generated, display information about the frequency of each number.

```
import random

# The main function.
def main():

    # Initialize an empty dictionary.
    number_dict = dict()

    # Repeat 100 times.
    for i in range(100):
        # Generate a random number between 1 and 10.
        random_number = random.randint(1, 10)

        # Establish or increment the number in the dictionary.
        if random_number not in number_dict:
            number_dict[random_number] = 1    #Finding it for the first time
        else:
            number_dict[random_number] += 1   #Increase value by one

    # Display the results.
    print('Number\tFrequency')
    print('------  ---------')

    # The "sorted" function produces a sorted version of
    # the list of key-value pairs from the "items" method.
    for number, frequency in sorted(number_dict.items()):
        print(format(number,'6d'), format(frequency,'10d') )

# Call the main function.
main()
```

Program Output

```
Number  Frequency
------  ---------
     1         11
     2         10
     3          8
     4          6
     5         10
     6         18
     7          9
     8         12
     9          8
    10          8
```

Review Questions

Page 502-503

Multiple Choice

1. You can use the _____ operator to determine whether a key exists in a dictionary.
   a. &
   b. in
   c. ^
   d. ?
2. You use _____ to delete an element from a dictionary.
   a. the `remove` method
   b. the `erase` method
   c. the `delete` method
   d. the `del` statement
3. The _____ function returns the number of elements in a dictionary:
   a. `size()`
   b. `len()`
   c. `elements()`
   d. `count()`
4. You can use _____ to create an empty dictionary.
   a. `{}`
   b. `()`
   c. `[]`
   d. `empty()`
5. The _____ method returns a randomly selected key-value pair from a dictionary.
   a. `pop()`
   b. `random()`
   c. `popitem()`
   d. `rand_pop()`

Sets and Creating and Empty Set Page 484

- **<u>Set</u>: object that stores a collection of data in same way as mathematical set**
  - All items must be unique – No duplicates in a set
  - Set is unordered – not sorted
  - Elements can be of different data types

- **<u>`set` function</u>: used to create a set**
  - For empty set, call `set()`

```
>>> myset = set()
>>> print( myset )
set()
```

Creating a Non-Empty Set

- **Set function is a one-argument function.**

  `myset=set( 'a' )`

  `myset=(17)` ✕ *Single Integer, Python 3 does not allow*

- **For non-empty set, call `set(`*`argument`*`)` where *`argument`* is an object that contains iterable elements**
  - **e.g., *`argument`* can be a list, string, or tuple**
  - **If *`argument`* is a string, each character becomes a set element**

    `myset=set( 'abc' )`

    - **For set of strings, pass them to the function as a list**

  myset=set('one' , 'two' , 'three') => TypeERROR

  `myset=set( [ 'one', 'two', 'three' ] )`

  - **If *`argument`* contains duplicates, only one of the duplicates will appear in the set**

Getting the Number of Elements in a Set

- **`len` function: returns the number of elements in the set**

```
>>> myset = set('a')
>>> len(myset)
1

>>> myset = set('Ali')
>>> len(myset)
3

>>> myset = set(['Ali','Ahmet'])
>>> len(myset)
2

>>> myset = set([1, 2, 3, 4, 5])
>>> len(myset)
5

>>> myset = set()
>>> len(myset)
0
```

Adding Elements 1/2

- **Sets are mutable objects**
- **<u>add method</u>: adds an element to a set**

```
>>> myset = set()
>>> myset.add(1)
>>> len(myset)
1
>>> myset.add(2)
>>> myset.add(3)
>>> myset
{1, 2, 3}
```

```
>>> myset.add(4)
>>> myset.add(2)
>>> myset
{1, 2, 3, 4}
>>> len(myset)
4
```

No Duplicate Elements in a Set!
2 is already exist, so it is skipped.

Adding Elements 2/2

- **Sets are mutable objects – possible to add multiple elements**
- **`update` method: adds a group of elements to a set**
  - Argument must be a sequence containing iterable elements, and each of the elements is added to the set

```
>>> myset = set([1, 2, 3])
>>> myset.update([4, 5, 6])
>>> myset
{1, 2, 3, 4, 5, 6}
>>>
>>> myset.update([3, 5, 7, 9])
>>> myset
{1, 2, 3, 4, 5, 6, 7, 9}
```

3 is already exist, so it is skipped/ignored.

```
>>> set1 = set([1, 3, 5, 7, 9])
>>> set2 = set([2, 4, 6, 8, 10])
>>> set1.update(set2)
>>> set1
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>>
>>> set1.add(0)
>>> set1
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Deleting Elements From a Set 1/2

- **`remove` and `discard` methods: remove the specified item from the set**
  - The item that should be removed is passed to both methods as an argument
  - Behave differently when the specified item is not found in the set
    - `remove` method raises a `KeyError` exception
    - `discard` method does not raise an exception

```
>>> myset = set([ 1, 2, 3, 4, 5 ])
>>> myset.add('a')
>>> myset.update('Chicago')
>>> myset
{1, 2, 3, 4, 5, 'o', 'g', 'c', 'a', 'i', 'C', 'h'}
>>> myset.remove(1)
>>> myset.discard(2)
>>> myset
{3, 4, 5, 'o', 'g', 'c', 'a', 'i', 'C', 'h'}
>>> myset.discard('i')
>>> myset.remove('a')
>>> myset
{3, 4, 5, 'o', 'g', 'c', 'C', 'h'}
>>> myset.discard(99)
>>> myset.remove(99)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    myset.remove(99)
KeyError: 99
```

Deleting Elements From a Set 2/2

- **`clear` method: clears all the elements of the set**

```
>>> myset
{3, 4, 5, 'o', 'g', 'c', 'C', 'h'}
>>> myset.clear()
>>> myset
set()
```

Using the `for` Loop with a Set

- **A `for` loop can be used to iterate over elements in a set**
  - General format:

```
for var in set:
    statement
    statement
    etc.
```

  - The loop iterates once for each element in the set

```
>>> myset = set([ 1, 2, 3, 4, 5 ])
>>> for item in myset:
        print(item)
```

```
1
2
3
4
5
```

```
>>> cities = set([ 'Adana', 'Gaziantep', 'Istanbul', 'Ankara' ])
>>> for city in cities:
        print(city)


Istanbul
Ankara
Adana
Gaziantep
```

`in` and `not in` Operators with a Set

- **The `in` operator can be used to test whether a value exists in a set**

```
>>> myset = set([1, 2, 3])
>>> if 1 in myset:
        print('The value 1 is in the set.')

The value 1 is in the set.
```

  - Similarly, the `not in` operator can be used to test whether a value does not exist in a set

```
>>> names=set()
>>> names.add('John')
>>> names.add('Taylor')
>>> names
{'Taylor', 'John'}
>>> if 'Talor' not in names:
        print('Talor is not in names')


Talor is not in names
```

Finding the Union of Sets

- **<u>Union of two sets</u>: a set that contains all the elements of both sets – combination of elements is the union of both-**
- **To find the union of two sets:**
  - Use the `union` method
    - Format: *`set1`*`.union(`*`set2`*`)`
  - Use the `|` operator
    - Format: *`set1 | set2`*
  - Both techniques return a new set which contains the union of both sets

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([3, 4, 5, 6])
>>> print(set1.union(set2))
{1, 2, 3, 4, 5, 6}
>>> set3 = set1.union(set2)
>>> set4 = set1 | set2
>>> set3
{1, 2, 3, 4, 5, 6}
>>> set4
{1, 2, 3, 4, 5, 6}
```

Finding the Intersection of Sets

- **<u>Intersection of two sets</u>: a set that contains only the elements found in both sets – *common elements*-**
- **To find the intersection of two sets:**
  - Use the `intersection` method
    - Format: *`set1`*`.intersection(`*`set2`*`)`
  - Use the `&` operator
    - Format: *`set1 & set2`*
  - Both techniques return a new set which contains the intersection of both sets

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([3, 4, 5, 6])
>>> set1.intersection(set2)
{3, 4}
>>> set2.intersection(set1)
{3, 4}
>>> set2 & set1
{3, 4}
>>> set1 & set2
{3, 4}
```

Finding the Difference of Sets

- **<u>Difference of two sets</u>: a set that contains the elements that appear in the first set but do not appear in the second set**
- **To find the difference of two sets:**
  - Use the `difference` method
    - Format: *`set1`*`.difference(`*`set2`*`)`
  - Use the – operator
    - Format: *`set1 - set2`*

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([3, 4, 5, 6])
>>> set1.difference(set2)
{1, 2}
>>> set2.difference(set1)
{5, 6}
>>> set1-set2
{1, 2}
>>> set2-set1
{5, 6}
```

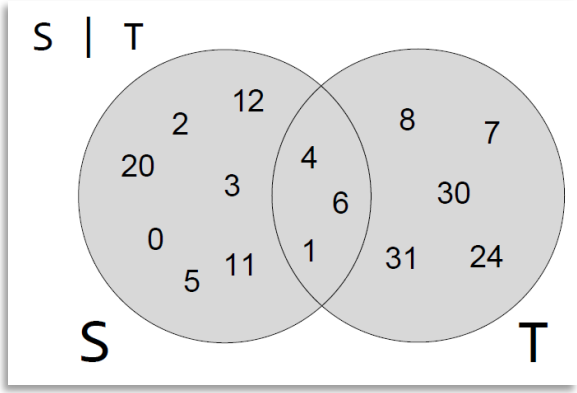
Finding the Symmetric Difference of Sets

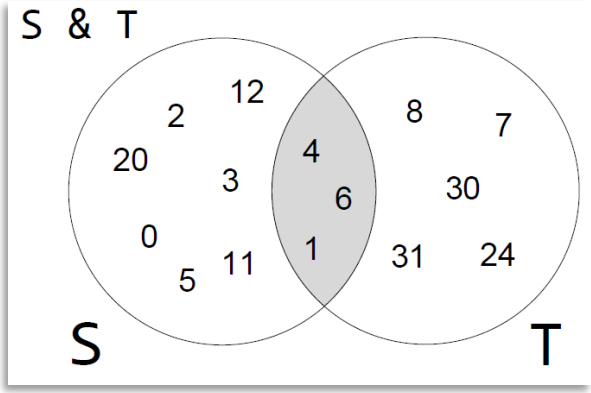
- **<u>Symmetric difference of two sets</u>: a set that contains the elements that are not shared by the two sets**
- **To find the symmetric difference of two sets:**
  - Use the `symmetric_difference` method
    - Format: *`set1`*`.symmetric_difference(`*`set2`*`)`
  - Use the ^ operator
    - Format: *`set1 ^ set2`*

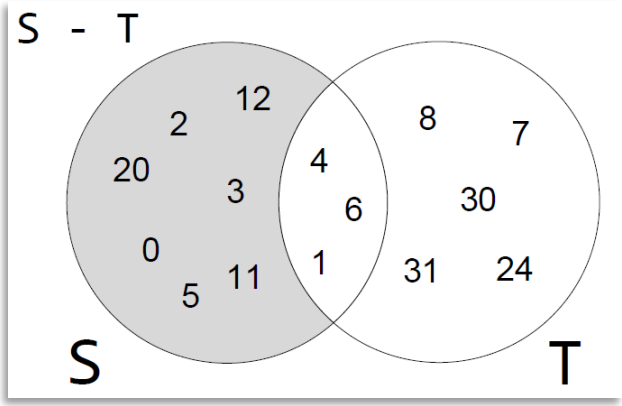
```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([3, 4, 5, 6])
>>> set1.symmetric_difference(set2)
{1, 2, 5, 6}
>>> set2.symmetric_difference(set1)
{1, 2, 5, 6}
>>> set1 ^ set2
{1, 2, 5, 6}
>>> set2 ^ set1
{1, 2, 5, 6}
```

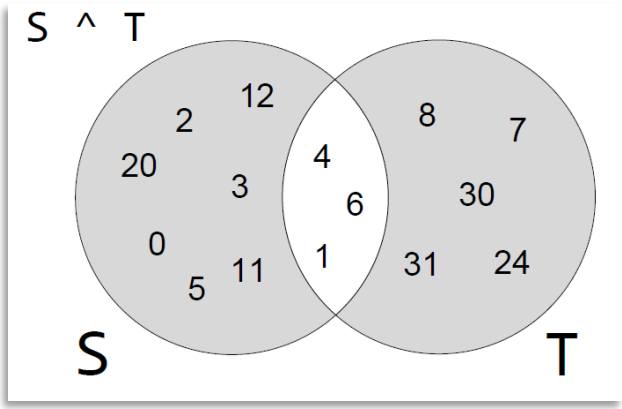
Example: | , & , - , and ^ set operators

```
S = {0, 1, 2, 3, 4, 5, 6, 11, 12, 20}
T = {1, 4, 6, 7, 8, 24, 31}
```









Finding Subsets

- **Set A is subset of set B if all the elements in set A are included in set B or vice versa.**
- **To determine whether set A is subset of set B**
  - Use the `issubset` method
    - Format: *setA*`.issubset(`*setB*`)`
  - Use the <= operator
    - Format: *setA* `<=` *setB*

```
>>> set1 = set([1, 2, 3, 4, 5, 6])
>>> set2 = set([2, 3, 4, 5])
>>> set2.issubset(set1)
True
>>> set1.issubset(set2)
False
>>>
>>> set3 = set([3, 4, 5])
>>> set3.issubset(set1)
True
>>> set3.issubset(set2)
True
```

```
>>> set2 <= set1
True
>>> set1 <= set2
False
```

Finding Supersets

- **Set A is superset of set B if it contains all the elements of set B**
- **To determine whether set A is superset of set B**
  - Use the `issuperset` method
    - Format: *`setA`*`.issuperset(`*`setB`*`)`
  - Use the >= operator
    - Format: *`setA >= setB`*

```
>>> set1 = set([ 1, 2, 3, 4, 5, 6])
>>> set2 = set([ 3, 4, 5, 6])
>>> set3 = set([ 4, 5])
>>> set1.issuperset(set2)
True
>>> set1 >= set2
True
>>> set2.issuperset(set3)
True
>>> set2.issuperset(set1)
False
```

9.15 Does a set allow you to store duplicate elements?

No, sets don't allow us to store duplicate elements. Elements must be different.

9.16 How do you create an empty set?

`set()` creates an empty set

9.17 After the following statement executes, what elements will be stored in the

```
myset set?
myset = set('Jupiter')
```

`{'r', 'i', 'e', 'J', 't', 'p', 'u'}` – Characters in Jupiter!

9.18 After the following statement executes, what elements will be stored in the `myset` set?

```
myset = set(25)
```

This is not possible in Python 3! Only iterable objects can be stored, this is only an integer value.

9.19 After the following statement executes, what elements will be stored in the `myset` set?

```
myset = set('www xxx yyy zzz')
```

`{'w', ' ', 'z', 'x', 'y'}`

9.21 After the following statement executes, what elements will be stored in the `myset` set?

```
myset = set(['www', 'xxx', 'yyy', 'zzz'])
```

`{'www', 'zzz', 'yyy', 'xxx'}` – each string is stored!

9.27 After the following code executes, what elements will be members of set3?

```
set1 = set([10, 20, 30])
set2 = set([100, 200, 300])
set3 = set1.union(set2)
```

{20, 100, 200, 10, 300, 30}

9.28 After the following code executes, what elements will be members of set3?

```
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1.intersection(set2)
```

{3, 4}

9.31 After the following code executes, what elements will be members of set3?

```
set1 = set(['a', 'b', 'c'])
set2 = set(['b', 'c', 'd'])
set3 = set1.symmetric_difference(set2)
```

{'d', 'a'}

9.32 Look at the following code:

```
set1 = set([1, 2, 3, 4])
set2 = set([2, 3])
```

Which of the sets is a subset of the other?

Which of the sets is a superset of the other?

set1 is superset to set2 set1 <= set2

set2 is subset to set1 set2 >= set1

In the Spotlight:

Set Operations

In this section, you will look at Program 9-3, which demonstrates various set operations. The program creates two sets: one that holds the names of students on the baseball team, and another that holds the names of students on the basketball team. The program then performs the following operations:

- It finds the intersection of the sets to display the names of students who play both sports.
- It finds the union of the sets to display the names of students who play either sport.
- It finds the difference of the baseball and basketball sets to display the names of students who play baseball but not basketball.
- It finds the difference of the basketball and baseball (*basketball – baseball*) sets to display the names of students who play basketball but not baseball. It also finds the difference of the baseball and basketball (*baseball – basketball*) sets to display the names of students who play baseball but not basketball.
- It finds the symmetric difference of the basketball and baseball sets to display the names of students who play one sport but not both.

Program 9-3 (sets.py)

```
# This program demonstrates various set operations.
baseball = set(['Jodi', 'Carmen', 'Aida', 'Alicia'])
basketball = set(['Eva', 'Carmen', 'Alicia', 'Sarah'])

```

Review the Spotlight Problem 9.3 for an application of sets in programming!

Serializing Objects

- **<u>Serialize an object</u>: convert the object to a stream of bytes that can easily be stored in a file**

In Chapter 6, you learned how to store data in a text file. Sometimes you need to store the contents of a complex object, such as a dictionary or a set, to a file. The easiest way to save an object to a file is to serialize the object. When an object is *serialized*, it is converted to a stream of bytes that can be easily stored in a file for later retrieval.

- **<u>Pickling</u>: serializing an object is called pickling**

Pickling Object/s – `pickle.dump`

- **To pickle an object into a binary file:**
  - Import the `pickle` module

    ```
    >>> import pickle
    ```

  - Open a file for binary writing – use `wb` mode

    ```
    file_object =  open('filename.dat','wb')
    ```

  - Call the `pickle.dump` function
    - Format: `pickle.dump(`*`object, file_object`*`)`
  - Here object to be pickled could be a dictionary or set
  - Close the file
- **You can pickle multiple objects to one file prior to closing the file**

Example: Pickling a Dictionary into a file

Let's write a program that pickles/writes a dictionary into a file (let's pickle the phonebook dictionary).

```
# This program pickles a dictionary into a Binary File
def main():
    import pickle
    phonebook = {'Chris' : '555-1111',
                 'Katie' : '555-2222',
                 'Joanne' : '555-3333'}
    output_file = open('phonebook.dat', 'wb')
    pickle.dump(phonebook, output_file)
    output_file.close()
# Calling Main Function
main()
```

After Running Program

phonebook.dat

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 80 03 7D 71 00 28 58 05 00 00 00 43 68 72 69 73 ; €.}q.(X....Chris
00000010h: 71 01 58 08 00 00 00 35 35 35 2D 31 31 31 31 71 ; q.X....555-1111q
00000020h: 02 58 05 00 00 00 4B 61 74 69 65 71 03 58 08 00 ; .X....Katieq.X..
00000030h: 00 00 35 35 35 2D 32 32 32 32 71 04 58 06 00 00 ; ..555-2222q.X...
00000040h: 00 4A 6F 61 6E 6E 65 71 05 58 08 00 00 00 35 35 ; .Joanneq.X....55
00000050h: 35 2D 33 33 33 33 71 06 75 2E                   ; 5-3333q.u.
```

Unpickling Object/s – `pickle.load`

- **<u>Unpickling</u>: retrieving pickled object from a binary file**
- **To unpickle an object:**
  - Import the `pickle` module

    ```
    >>> import pickle
    ```

  - Open a file for binary reading with `rb` mode

    ```
    file_object =  open('filename.dat','rb')
    ```

  - Call the `pickle.load` function for unpickling
    - Format: `dict = pickle.load(file_object)`
  - Close the file
- **You can unpickle multiple objects from the file**

Example: Unpickling a Dictionary from a file

Let's now write a program that unpickles data from `phonebook.dat` file created in the previous example.

```
# This program unpickles a dictionary from a Binary File
# Also displays the content in seprate lines
def main():
     import pickle
     input_file = open('phonebook.dat', 'rb')
     # Load the dictionary into pb dictionary
     pb=pickle.load(input_file)
     # Iterate through the keys in the pb dictionary
     for k in pb:
          print( k,':', pb[k] )
     #Close the file
     input_file.close()
# Calling Main Function
main()
```

Program Output

```
Chris : 555-1111
Katie : 555-2222
Joanne : 555-3333
```

9.33 What is object serialization?
Convert an object to a stream of bytes that can easily be stored in a file

9.34 When you open a file for the purpose of saving a pickled object to it, what file access mode do you use?
`wb` – Binary Write Mode

9.35 When you open a file for the purpose of retrieving a pickled object from it, what file access mode do you use?
`rb` – Binary Read Mode

9.36 What module do you import if you want to pickle objects?
`pickle` module is needed to be imported.

9.37 What function do you call to pickle an object?
`pickle.dump` function is used to pickle an object to a file

9.38 What function do you call to retrieve and unpickle an object?
`pickle.load` function is used to unpickle an object from a file

Review Questions

Page 503-504

Multiple Choice

11. You can add a group of elements to a set with this method.
    a. `append`
    b. `add`
    c. `update`
    d. `merge`

13. This `set` method removes an element and raises an exception if the element is not found.
    a. `remove`
    b. `discard`
    c. `delete`
    d. `erase`

15. This operator can be used to find the difference of two sets.
    a. `|`
    b. `&`
    c. `-`
    d. `^`

17. This operator can be used to find the symmetric difference of two sets.
    a. `|`
    b. `&`
    c. `-`
    d. `^`

Algorithm Workbench

6. Assume each of the variables `set1` and `set2` references a set. Write code that creates another set containing all the elements of `set1` and `set2`, and assigns the resulting set to the variable `set3`.

```
set3 = set1.union(set2)  or set3 = set1 | set2
```

9. Assume each of the variables `set1` and `set2` references a set. Write code that creates another set containing the elements that appear in `set2` but not in `set1`, and assigns the resulting set to the variable `set3`.

```
set3 = set2.difference(set1)  or set3 = set2 - set1
```

10. Assume that `set1` references a set of integers and `set2` references an empty set. Write code that iterates through each element of `set1`. If the element is greater than 100, add it to `set2`.

```
for item in set1:
    if item > 100:
        set2.add(item)
```

11. Assume the variable `dct` references a dictionary. Write code that pickles the dictionary and saves it to a file named `mydata.dat`.

```
output = open( 'mydata.dat','wb' )
pickle.dump(dct, output)
```

Programming Exercises

4. Unique Words: Write a program that opens a specified text file then displays a list of all the unique words found in the file.

Hint: Store each word as an element of a set.

text.txt - Notepad

File Edit Format View Help

```
No one is unaware of the name of that famous English shipowner, Cunard.
In 1840 this shrewd industrialist founded a postal service between Liverpool and Halifax, featuring three wooden sh
Eight years later, the company's assets were increased by four 650-horsepower ships at 1,820 metric tons, and in tw
In 1853 the Cunard Co., whose mail-carrying charter had just been renewed, successively added to its assets the Ara
So in 1867 this company owned twelve ships, eight with paddle wheels and four with propellers.
If I give these highly condensed details, it is so everyone can fully understand the importance of this maritime tr
No transoceanic navigational undertaking has been conducted with more ability, no business dealings have been crown
In twenty-six years Cunard ships have made 2,000 Atlantic crossings without so much as a voyage canceled, a delay r
Accordingly, despite strong competition from France, passengers still choose the Cunard line in preference to all o
Given this, no one will be astonished at the uproar provoked by this accident involving one of its finest steamers.
```

Programming Exercises

4. Unique Words Program

```
# Program reads a text file and finds the unique words
# Note that input file here is entered by the user
def main():
    # Get name of input file.
    input_name = input('Enter the name of the input file: ')

    # Open the input file and read the text.
    input_file = open(input_name, 'r')
    text = input_file.read() #read the whole content
    words = text.split()  # split by space and store in a list
    unique_words = set()  # Creating empty set to store unique words

    # Processing the list of the words/removing,.!?.
    for word in words:
        word=word.replace(',','')
        word=word.replace('!','')
        word=word.replace('?','')
        word=word.replace('.','')
        word=word.lower()  #converting each word to lower case
        unique_words.add(word)
    # Printing the Unique words and number of words
    for word in unique_words:
        print(word)
    print('Total # of words before elimination:',len(words))
    print('There are',len(unique_words),'unique words.)

    # Close the file.
    input_file.close()

# Call the main function.
main()
```

```
Enter the name of the input file: text.txt
2000
steamers
business
seen
these
at
.
.
.
if
liverpool
Total # of words before elimination: 270
There are 169 unique words.
```

Summary

- **Dictionaries, including:**
  - Creating dictionaries
  - Inserting, retrieving, adding, and deleting key-value pairs
  - `for` loops and `in` and `not in` operators
  - Dictionary methods
- **Sets:**
  - Creating sets
  - Adding elements to and removing elements from sets
  - Finding set union, intersection, difference and symmetric difference
  - Finding subsets and supersets
- **Serializing objects:**
  - Pickling and unpickling objects