CN101

Lecture 8

Dictionaries

Dictionaries

- Dictionary: 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 an immutable object
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

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

Creating a Dictionary

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

- In this example, the keys and the values are strings. The values in a dictionary can be objects of any type, but the keys must be immutable objects.
- For example, keys can be strings, integers, floating-point values, or tuples.
- Keys cannot be lists or any other type of immutable object.

Dictionaries in Python 3.7 and Later

- Starting from Python 3.7, dictionaries are guaranteed to preserve the order of insertion. This means that when you iterate over a dictionary, the items are returned in the order they were added.
- If you are using Python 3.6 or earlier, dictionaries are unordered, meaning they do not maintain any specific order of the items.

```
>>> # Creating a dictionary in Python 3.7+
>>> my_dict = {'first': 1, 'second': 2, 'third': 3}
>>> my_dict
{'first': 1, 'second': 2, 'third': 3}
>>>
>>> # Adding more items
>>> my_dict['fourth'] = 4
>>> my_dict['fifth'] = 5
>>> my_dict
{'first': 1, 'second': 2, 'third': 3, 'fourth': 4, 'fifth': 5}
>>>
>>> # Iterating over the dictionary
>>> for key, value in my_dict.items():
...     print(key, value)
...
first 1
second 2
third 3
fourth 4
fifth 5
```

Retrieving a Value from a Dictionary

- General format for retrieving value from dictionary: `dictionary[key]`
  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised
- Test whether a key is in a dictionary using the `in` and `not in` operators
  - Helps prevent `KeyError` exceptions

Retrieving a Value from a Dictionary (cont'd.)

```
>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222',
'Joanne':'555-3333'}
>>> phonebook['Chris']
'555-1111'
>>> phonebook['Katie']
'555-2222'
>>> phonebook['Joanne']
'555-3333'
>>> phonebook['peter']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'peter'
```

Using the `in` and `not in` Operators to Test for a Value in a Dictionary

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

Using the `in` and `not in` Operators to Test for a Value in a Dictionary (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...              'Katie':'555-2222',
...              'Joanne':'555-3333'}
>>> if 'Somsak' not in phonebook:
...     print('Somsak is not found.')
...
Somsak is not found.
>>>
```

Adding Elements to an Existing Dictionary

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

```
dictionary[key] = value
```

  - If key exists in the dictionary, the value associated with it will be changed

Adding Elements to an Existing Dictionary (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...              'Katie':'555-2222',
...              'Joanne':'555-3333'}
>>> phonebook['Somsak'] = '02-564-3001'
>>> phonebook['Chris'] = '555-4444'
>>> phonebook
{'Chris': '555-4444', 'Katie': '555-2222',
'Joanne': '555-3333', 'Somsak': '02-564-3001'}
>>>
```

Deleting Elements From an Existing Dictionary

- To delete a key-value pair:

```
del dictionary[key]
```

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

Deleting Elements From an Existing Dictionary (cont'd.)

```
>>> phonebook = {'Chris': '555-4444', 'Katie': '555-2222',
'Joanne': '555-3333', 'Somsak': '02-564-3001'}
>>> if 'Somsak' in phonebook:
...     del phonebook['Somsak']
...
>>> phonebook
{'Chris': '555-4444', 'Katie': '555-2222', 'Joanne':
'555-3333'}
>>> del phonebook['Somsak']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Somsak'
```

Getting the Number of Elements

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

```
>>> phonebook = {'Chris': '555-4444', 'Katie': '555-2222'}
>>> num_items = len(phonebook)
>>> num_items
2
>>>
```

Mixing Data Types

- Keys must be immutable objects, but associated values can be any type of object
  - One dictionary can include keys of several different immutable types
- Values stored in a single dictionary can be of different types

Mixing Data Types (cont'd.)

```
>>> test_scores = { 'Clark': [88, 92, 100],
...                 'Lois': [95, 74, 21],
...                 'Lex': [72, 88, 91]}
>>> test_scores
{'Clark': [88, 92, 100], 'Lois': [95, 74, 21],
'Lex': [72, 88, 91]}
>>> test_scores['Clark']
[88, 92, 100]
>>>
```

Mixing Data Types (cont'd.)

```
>>> mixed_up = {'abc':1,
...                 999:'yada yada',
...                 (3, 6, 9):[3, 6, 9]}
>>> mixed_up
{'abc': 1, 999: 'yada yada', (3, 6, 9): [3, 6, 9]}
>>>
```

Mixing Data Types (cont'd.)

```
>>> employee = {'name' : 'Kevin Smith',
...              'id' : 12345,
...              'payrate' : 25.75 }
>>> employee
{'name': 'Kevin Smith', 'id': 12345, 'payrate': 25.75}
>>>
```

Creating an Empty Dictionary

- To create an empty dictionary:
  - Use `{}`
  - Use built-in function `dict()`
  - Elements can be added to the dictionary as program executes

Creating an Empty Dictionary (cont'd.)

```
>>> phonebook = {}
>>> phonebook['Chris'] = '555-1111'
>>> phonebook['Katie'] = '555-2222'
>>> phonebook
{'Chris': '555-1111', 'Katie': '555-2222'}
>>>
```

Using `for` Loop to Iterate Over a Dictionary

- Use a `for` loop to iterate over a dictionary
  - General format: `for` *`key`* `in` *`dictionary`*`:`

```
for key in dictionary:
   statement
   statement
   etc.
```

Using `for` Loop to Iterate Over a Dictionary (cont'd.)

```
>>> employee = {'name' : 'Kevin Smith',
...              'id' : 12345,
...              'payrate' : 25.75 }
>>> for key in employee:
...     print(key)
...
name
id
payrate
```

Using `for` Loop to Iterate Over a Dictionary (cont'd.)

```
>>> employee = {'name' : 'Kevin Smith',
...              'id' : 12345,
...              'payrate' : 25.75 }
>>> for key in employee:
...     print(f'{key} : {employee[key]}')
...
name : Kevin Smith
id : 12345
payrate : 25.75
```

Some Dictionary Methods

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.
values	Returns all the values in the 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.
update	Updates the dictionary with key-value pairs from another dictionary or from an iterable of key-value pairs.

Some Dictionary Methods

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

```
>>> phonebook = {'Chris':'555-1111',
...                'Katie':'555-2222'}
>>> phonebook
{'Chris': '555-1111', 'Katie': '555-2222'}
>>> phonebook.clear()
>>> phonebook
{}
```

Some Dictionary Methods (cont'd.)

- `get` method: gets a value associated with specified key from the dictionary
  - Format: *`dictionary`*`.get(`*`key`*`, `*`default`*`)`
    - *`default`* is returned if *`key`* is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

Some Dictionary Methods (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...                'Katie':'555-2222'}
>>> phonebook.get('Chris')
'555-1111'
>>> phonebook.get('Chri')
>>> phonebook.get('Chri', 'Entry not found')
'Entry not found'
>>>
```

Some Dictionary Methods (cont'd.)

- `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

Some Dictionary Methods (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...              'Katie':'555-2222'}
>>> phonebook.items()

dict_items([('Chris', '555-1111'), ('Katie', '555-2222')])


>>> for key, value in phonebook.items():
...     print(f'{key} : {value}')
...
Chris : 555-1111
Katie : 555-2222
```

Some Dictionary Methods (cont'd.)

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

Some Dictionary Methods (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...                'Katie':'555-2222'}
>>> phonebook.keys()
dict_keys(['Chris', 'Katie'])
>>>
>>> for key in phonebook.keys():
...     print(key)
...
Chris
Katie
```

Some Dictionary Methods (cont'd.)

- `values` method: returns all the dictionaries values as a sequence
  - Format: *`dictionary`*`.values()`
  - Use a `for` loop to iterate over the values

Some Dictionary Methods (cont'd.)

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

Example: book_inventory.py

- This program manages a book inventory and allows users to check out books. The `inventory` dictionary keeps track of book titles and their available quantities.
- The `checkout_book` function reduces the quantity of a specified book if it's available.
- The `display_inventory` function shows all books and their quantities in a numbered list.
- The `main` function provides a loop for users to view the inventory and check out books until they choose to quit.

book_inventory.py (Chapter 09)

```
# Initial book inventory
inventory = {
  "The Catcher in the Rye": 4,
  "To Kill a Mockingbird": 2,
  "1984": 5,
  "The Great Gatsby": 3
}

# Function to check out a book
def checkout_book(title):
  # Use get to avoid KeyError if the book is not in the inventory
  available_copies = inventory.get(title)
  if available_copies > 0:
    inventory[title] -= 1
    print(f"Checked out '{title}'. Remaining copies: {inventory[title]}")
  else:
    print(f"Sorry, '{title}' is not available.")

# Function to display all books and their quantities
def display_inventory():
  print("Current book inventory:")
  for id, (title, quantity) in enumerate(inventory.items(), start=1):
    print(f"{id}. {title}: {quantity} copies")

```

book_inventory.py (cont'd.)

```
25 def main():
26   while True:
27     # Display the current inventory
28     display_inventory()
29
30     # Prompt user for a book to check out
31     book_number = input("Enter the number of the book to check out (or
   'quit' to exit): ").strip()
32
33     if book_number.lower() == 'quit':
34       print("Exiting the program.")
35       break
36
37     if not book_number.isdigit() or int(book_number) < 1 or
   int(book_number) > len(inventory):
38       print("Invalid input. Please enter a valid book number.")
39       continue
40
41     book_index = int(book_number) - 1
42     book_title = list(inventory.keys())[book_index]
43
```

book_inventory.py (cont'd.)

```
44      # Attempt to check out the book
45      checkout_book(book_title)
46
47  main()
```

Program Output

```
Current book inventory:
1. The Catcher in the Rye: 4 copies
2. To Kill a Mockingbird: 2 copies
3. 1984: 5 copies
4. The Great Gatsby: 3 copies
Enter the number of the book to check out (or 'quit' to exit): 2
Checked out 'To Kill a Mockingbird'. Remaining copies: 1
Current book inventory:
1. The Catcher in the Rye: 4 copies
2. To Kill a Mockingbird: 1 copies
3. 1984: 5 copies
4. The Great Gatsby: 3 copies
Enter the number of the book to check out (or 'quit' to exit): 5
Invalid input. Please enter a valid book number.
```

Program Output (cont'd.)

```
Current book inventory:
1. The Catcher in the Rye: 4 copies
2. To Kill a Mockingbird: 1 copies
3. 1984: 5 copies
4. The Great Gatsby: 3 copies
Enter the number of the book to check out (or 'quit' to exit): 2
Checked out 'To Kill a Mockingbird'. Remaining copies: 0
Current book inventory:
1. The Catcher in the Rye: 4 copies
2. To Kill a Mockingbird: 0 copies
3. 1984: 5 copies
4. The Great Gatsby: 3 copies
Enter the number of the book to check out (or 'quit' to exit): 2
Sorry, 'To Kill a Mockingbird' is not available.
Current book inventory:
1. The Catcher in the Rye: 4 copies
2. To Kill a Mockingbird: 0 copies
3. 1984: 5 copies
4. The Great Gatsby: 3 copies
Enter the number of the book to check out (or 'quit' to exit): quit
Exiting the program.
```

Some Dictionary Methods (cont'd.)

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

Some Dictionary Methods (cont'd.)

```
>>> phonebook = {'Chris':'555-1111',
...              'Katie':'555-2222'}
>>> phone_num = phonebook.pop('Chris', 'Entry not found')
>>> phone_num
'555-1111'
>>> phonebook
{'Katie': '555-2222'}
>>> phone_num = phonebook.pop('Chris', 'Entry not found')
>>> phone_num
'Entry not found'
```

Some Dictionary Methods (cont'd.)

- `popitem` method: removes the last inserted key-value pair from the dictionary and returns it as a tuple.
  - Format: *`dictionary`*`.popitem()`
  - Key-value pair returned as a tuple

Some Dictionary Methods (cont'd.)

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

Example: to-do_list.py

- This program manages a to-do list, allowing users to display tasks, mark tasks as completed, and remove them from the list. The `todo_list` dictionary stores tasks with their priorities and descriptions.
- The `print_todo_list` function displays all tasks, while the `complete_task` function prompts the user to complete a task and removes it from the list.
- The `main` function provides a menu for users to choose between displaying the list, completing a task, or exiting.

to-do_list.py

```
# Initial to-do list
todo_list = {
  'task1': {'priority': 'high', 'description': 'Finish the report'},
  'task2': {'priority': 'medium', 'description': 'Email the client'},
  'task3': {'priority': 'low', 'description': 'Update the project plan'},
  'task4': {'priority': 'high', 'description': 'Prepare for the meeting'}
}

def print_todo_list(todo_list):
  """Prints the to-do list."""
  print("\nCurrent To-Do List:")
  for task, details in todo_list.items():
    print(f"Task: {task}, Priority: {details['priority']}, Description: {details['description']}")

def complete_task(todo_list):
  """Allows user to mark a task as completed and remove it from the to-do list."""
  task = input("Enter the name of the task you have completed: ")
  details = todo_list.pop(task, None)
  if details:
    print(f"Completed and removed task: {task}, Details: {details}")
else:
    print(f"Task {task} not found in the to-do list.")

```

to-do_list.py (cont'd.)

```
24 def main():
25     while True:
26         print("\nTo-Do List Management Menu")
27         print("1. Display to-do list")
28         print("2. Mark task as completed")
29         print("3. Exit")
30         choice = input("Enter your choice (1-3): ")
31 
32         if choice == '1':
33             print_todo_list(todo_list)
34         elif choice == '2':
35             complete_task(todo_list)
36         elif choice == '3':
37             print("Exiting...")
38             break
39         else:
40             print("Invalid choice. Please try again.")
41 
42 main()
```

Program Output

```
To-Do List Management Menu
1. Display to-do list
2. Mark task as completed
3. Exit
Enter your choice (1-3): 1

Current To-Do List:
Task: task1, Priority: high, Description: Finish the report
Task: task2, Priority: medium, Description: Email the client
Task: task3, Priority: low, Description: Update the project plan
Task: task4, Priority: high, Description: Prepare for the meeting

To-Do List Management Menu
1. Display to-do list
2. Mark task as completed
3. Exit
Enter your choice (1-3): 2
Enter the name of the task you have completed: task2
Completed and removed task: task2, Details: {'priority': 'medium',
'description': 'Email the client'}
```

Program Output (cont'd.)

```
To-Do List Management Menu
1. Display to-do list
2. Mark task as completed
3. Exit
Enter your choice (1-3): 1

Current To-Do List:
Task: task1, Priority: high, Description: Finish the report
Task: task3, Priority: low, Description: Update the project plan
Task: task4, Priority: high, Description: Prepare for the meeting

To-Do List Management Menu
1. Display to-do list
2. Mark task as completed
3. Exit
Enter your choice (1-3): 3
Exiting...
```

Some Dictionary Methods (cont'd.)

- `update` method: updates the dictionary with key-value pairs from another dictionary or from an iterable of key-value pairs. This method adds key-value pairs from the provided dictionary or iterable to the original dictionary, overwriting existing keys if there are duplicates.
  - Format: *`dictionary`*`.update([other])`
    - `other`: Another dictionary or an iterable of key-value pairs (like a list of tuples) to update the original dictionary with. This parameter is optional.

Some Dictionary Methods (cont'd.)

Using another dictionary

```
>>> # Original dictionary
>>> dict1 = {'a': 1, 'b': 2}
>>> # Dictionary to update with
>>> dict2 = {'b': 3, 'c': 4}
>>>
>>> # Updating dict1 with dict2
>>> dict1.update(dict2)
>>>
>>> dict1
{'a': 1, 'b': 3, 'c': 4}
```

Some Dictionary Methods (cont'd.)

Using an iterable of key-value pairs

```
>>> # Original dictionary
>>> dict1 = {'a': 1, 'b': 2}
>>> # Iterable of key-value pairs to update with
>>> updates = [('b', 3), ('c', 4)]
>>>
>>> # Updating dict1 with the iterable
>>> dict1.update(updates)
>>>
>>> dict1
{'a': 1, 'b': 3, 'c': 4}
```

Some Dictionary Methods (cont'd.)

Using keyword arguments

```
>>> # Original dictionary
>>> dict1 = {'a': 1, 'b': 2}
>>>
>>> # Updating dict1 using keyword arguments
>>> dict1.update(b=3, c=4)
>>>
>>> dict1
{'a': 1, 'b': 3, 'c': 4}
```

user_profile.py

```python
def update_user_profile(user_profile, new_info):
    """
    Updates the user profile dictionary with new information.
    """
    user_profile.update(new_info)
    return user_profile

def get_user_input():
    """
    Collects updated user profile information from user input.
    """
    print("Enter updated profile information (leave blank to skip):")

    new_info = {}

    email = input("Email: ").strip()
    if email:
        new_info['email'] = email

    location = input("Location: ").strip()
    if location:
        new_info['location'] = location

```

user_profile.py (cont'd.)

```
    phone = input("Phone: ").strip()
    if phone:
        new_info['phone'] = phone

    return new_info

def main():
    # Initial user profile
    user_profile = {
        'username': 'johndoe',
        'email': 'johndoe@example.com',
        'first_name': 'John',
        'last_name': 'Doe',
        'location': 'New York'
    }

    # Get new information from user input
    new_info = get_user_input()

    # Update user profile with the new information
    updated_profile = update_user_profile(user_profile, new_info)

```

user_profile.py (cont'd.)

```
46      # Print the updated user profile
47      print("\nUpdated user profile:")
48      print(updated_profile)
49
50  main()
```

Program Output

```
Enter updated profile information (leave blank to skip):
Email: somsak@hotmail.com
Location: Bangkok
Phone: 025643001

Updated user profile:
{'username': 'johndoe', 'email': 'somsak@hotmail.com',
'first_name': 'John', 'last_name': 'Doe', 'location': 'Bangkok',
'phone': '025643001'}
```

Nested Dictionary

- A nested dictionary in Python is a dictionary that contains another dictionary as its value. This allows you to organize data in a hierarchical structure.

```
nested_dict = {
    'key1': {
        'subkey1': 'value1',
        'subkey2': 'value2'
    },
    'key2': {
        'subkey3': 'value3',
        'subkey4': 'value4'
    }
}
```

Nested Dictionary (cont'd.)

- To access values in a nested dictionary, you use a series of keys. Here's how you can access 'value1' from nested_dict:

```
>>> nested_dict = {
...     'key1': {
...         'subkey1': 'value1',
...         'subkey2': 'value2'
...     },
...     'key2': {
...         'subkey3': 'value3',
...         'subkey4': 'value4'
...     }
... }
>>> value = nested_dict['key1']['subkey1']
>>> value
'value1'
```

Nested Dictionary (cont'd.)

- You can modify values in a nested dictionary in a similar manner:

```
>>> nested_dict['key1']['subkey1'] = 'new_value1'
>>> nested_dict
{'key1': {'subkey1': 'new_value1', 'subkey2': 'value2'},
'key2': {'subkey3': 'value3', 'subkey4': 'value4'}}
```

Nested Dictionary (cont'd.)

- To add a new key-value pair to a nested dictionary:

```
>>> nested_dict['key1']['subkey3'] = 'value5'
>>> nested_dict
{'key1': {'subkey1': 'new_value1', 'subkey2':
'value2', 'subkey3': 'value5'}, 'key2': {'subkey3':
'value3', 'subkey4': 'value4'}}
```

```
>>> users = {
...     'alice': {
...         'age': 25,
...         'email': 'alice@example.com'
...     },
...     'bob': {
...         'age': 30,
...         'email': 'bob@example.com'
...     }
... }
>>>
>>> # Accessing Bob's age
>>> users['bob']['age']
30
>>>
>>> # Adding a new user
>>> users['charlie'] = {'age': 35, 'email':
'charlie@example.com'}
>>> users['charlie'])
{'age': 35, 'email': 'charlie@example.com'}
```

Example: company.py

- This program defines and uses a function called `print_company_data` to display information about employees within different departments of a company. It organizes and prints the data in a tabular format, showing each department and listing the employees' names, ages, positions, and emails.
- The `main` function initializes a nested dictionary representing the company's departments and employees, then calls `print_company_data` to print the structured data to the console.

company.py

```
1  def print_company_data(company):
2      """Prints the company data in a table format using nested loops."""
3      for department, employees in company.items():
4          print(f"\nDepartment: {department}")
5          print(f"{'Name':<10} {'Age':<5} {'Position':<20} {'Email':<25}")
6          for name, details in employees.items():
7              print(f"{name:<10}", end=" ")
8              for key, value in details.items():
9                  if key == 'age':
10                     print(f"{value:<5}", end=" ")
11                 elif key == 'position’:
12                     print(f"{value:<20}", end=" ")
13                 elif key == 'email’:
14                     print(f"{value:<25}", end=" ")
15             print() # for newline
16
17 def main():
18     company = {
19         'Engineering': {
20             'Alice': {'age': 29, 'position': 'Software Engineer', 'email':
               'alice@company.com'},
21             'Bob': {'age': 34, 'position': 'DevOps Engineer', 'email':
               'bob@company.com'}
22         },
```

company.py (cont'd.)


```
23          'HR': {
24              'Charlie': {'age': 40, 'position': 'HR Manager', 'email':
                'charlie@company.com'},
25              'Daisy': {'age': 28, 'position': 'Recruiter', 'email':
                'daisy@company.com'}
26          },
27          'Marketing': {
28              'Eve': {'age': 35, 'position': 'Marketing Specialist',
                'email': 'eve@company.com'},
29              'Frank': {'age': 31, 'position': 'SEO Expert', 'email':
                'frank@company.com'}
30          }
31      }
32      print_company_data(company)
33
34  main()
```

Program Output

```
Department: Engineering
Name        Age   Position              Email
Alice       29    Software Engineer     alice@company.com
Bob         34    DevOps Engineer       bob@company.com

Department: HR
Name        Age   Position              Email
Charlie     40    HR Manager            charlie@company.com
Daisy       28    Recruiter             daisy@company.com

Department: Marketing
Name        Age   Position              Email
Eve         35    Marketing Specialist eve@company.com
Frank       31    SEO Expert            frank@company.com
```

Example: users.py

- The overall functionality of this program is to manage user data stored in a dictionary. The program provides a command-line interface that allows the user to perform various operations on the user data, such as displaying, adding, updating, and deleting users.

users.py

```python
users = {
    'alice': {
        'age': 25,
        'email': 'alice@example.com'
    }
}

def print_users_table(users):
    """Prints the user data in a table format."""
    print(f"{'Name':<10} {'Age':<5} {'Email':<20}")
    for name, details in users.items():
        age = details['age']
        email = details['email']
        print(f"{name:<10} {age:<5} {email:<20}")

def create_user(users, name, age, email):
    """Create a new user to the dictionary."""
    if name in users:
        return False
    users[name] = {'age': age, 'email': email}
    return True

```

users.py (cont'd.)

```
23 def update_user(users, name, age=None, email=None):
24     """Updates an existing user in the dictionary."""
25     if name in users:
26         if age is not None:
27             users[name]['age'] = age
28         if email is not None:
29             users[name]['email'] = email
30         return True
31     else:
32         return False
33
34 def delete_user(users, name):
35     """Deletes a user from the dictionary."""
36     if name in users:
37         del users[name]
38         return True
39     else:
40         return False
41
42 def main():
43     while True:
44         print("\nUser Management Menu")
45         print("1. Display users")
46         print("2. Create user")
47         print("3. Update user")
```

users.py (cont'd.)

```
48          print("4. Delete user")
49          print("5. Exit")
50          choice = input("Enter your choice (1-5): ")
51
52          if choice == '1':
53              print("\nUsers table:")
54              print_users_table(users)
55          elif choice == '2':
56              name = input("Enter name: ")
57              age = int(input("Enter age: "))
58              email = input("Enter email: ")
59              if create_user(users, name, age, email):
60                  print(f"User {name} added.")
61              else:
62                  print(f"User {name} already exists.")
63          elif choice == '3':
64              name = input("Enter name: ")
65              age = input("Enter age (leave blank to skip): ")
66              email = input("Enter email (leave blank to skip): ")
67              age = int(age) if age else None
68              email = email if email else None
69              if update_user(users, name, age, email):
70                  print(f"User {name} updated.")
71              else:
72                  print(f"User {name} not found.")
```

users.py (cont'd.)

```
73          elif choice == '4':
74              name = input("Enter name: ")
75              if delete_user(users, name):
76                  print(f"User {name} deleted.")
77              else:
78                  print(f"User {name} not found.")
79          elif choice == '5':
80              print("Exiting...")
81              break
82          else:
83              print("Invalid choice. Please try again.")
84
85  main()
```

Program Output

```
User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 1
```

Program Output (cont'd.)

```
Users table:
Name        Age    Email
alice       25     alice@example.com

User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 2
Enter name: bob
Enter age: 30
Enter email: bob@example.com
User bob created.

User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
```

Program Output (cont'd.)

```
Enter your choice (1-5): 1

Users table:
Name        Age   Email
alice       25    alice@example.com
bob         30    bob@example.com

User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 3
Enter name: bob
Enter age (leave blank to skip): 40
Enter email (leave blank to skip): bob@hotmail.com
User bob updated.
```

Program Output (cont'd.)

```
User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 1

Users table:
Name        Age    Email
alice       25     alice@example.com
bob         40     bob@hotmail.com

User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 4
Enter name: alice
User alice deleted.
```

Program Output (cont'd.)

```
User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 1

Users table:
Name        Age    Email
bob         40     bob@hotmail.com

User Management Menu
1. Display users
2. Create user
3. Update user
4. Delete user
5. Exit
Enter your choice (1-5): 5
Exiting...
```

Summary

- This chapter covered:
  - Dictionaries, including:
    - Creating dictionaries
    - Inserting, retrieving, adding, and deleting key-value pairs
    - `for` loops and `in` and `not in` operators
    - Dictionary methods