Classes and Object-Oriented Programming

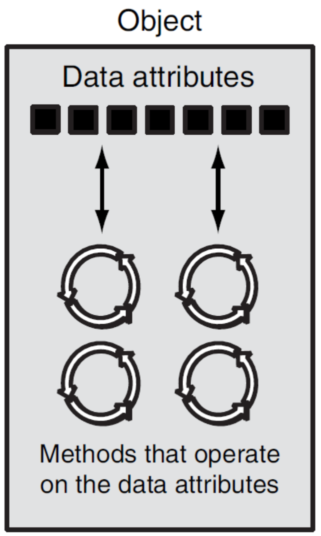
陳建良

Procedural and Object-Oriented Programming

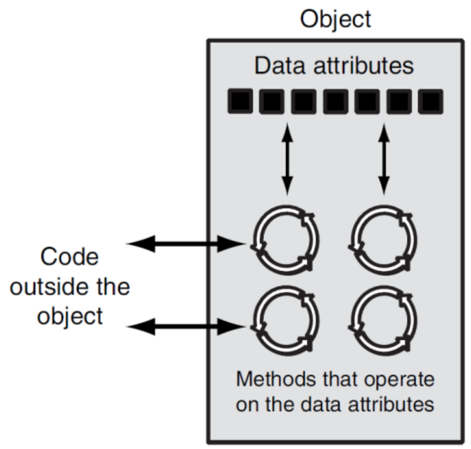
- There are primarily two methods of programming in use today: procedural and object-oriented.
- You can think of a *procedure* simply as a function that performs a specific task such as gathering input from the user, performing calculations, reading or writing files, displaying output, and so on.
- Whereas procedural programming is centered on creating procedures (functions), *object-oriented programming* (*OOP*) is centered on creating objects.
- An *object* is a software entity that contains both data and procedures.

Aletheia University
資訊工程學系

- The data contained in an object is known as the object's *data attributes.* An object's data attributes are simply variables that reference data.
- The procedures that an object performs are known as *methods.* An object's methods are functions that perform operations on the object's data attributes.
- The object is, conceptually, a self-contained unit that consists of data attributes and methods that operate on the data attributes.



- OOP addresses the problem of code and data separation through encapsulation and data hiding.
- *Encapsulation* refers to the combining of data and code into a single object.
- *Data hiding* refers to an object's ability to hide its data attributes from code that is outside the object.
- Only the object's methods may directly access and make changes to the object's data attributes.
- An object typically hides its data, but allows outside code to access its methods.



An Everyday Example of an Object

- Imagine that your alarm clock is actually a software object. If it were, it would have the following data attributes:
  - current_second (a value in the range of 0–59)
  - current_minute (a value in the range of 0–59)
  - current_hour (a value in the range of 1–12)
  - alarm_time (a valid hour and minute)
  - alarm_is_set (True or False)

- As you can see, the data attributes are merely values that define the *state* in which the alarm clock is currently.
- You, the user of the alarm clock object, cannot directly manipulate these data attributes because they are ***private***.
- To change a data attribute's value, you must use one of the object's methods.
- The following are some of the alarm clock object's methods:
  - set_time
  - set_alarm_time
  - set_alarm_on
  - set_alarm_off
- Methods that can be accessed by entities outside the object are known as ***public methods***.

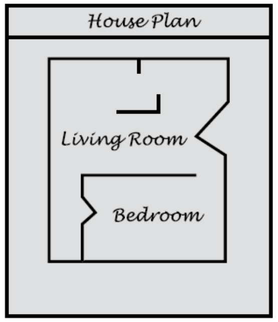
- The alarm clock also has *private methods*, which are part of the object's private, internal workings.
- External entities do not have direct access to the alarm clock's private methods.
- The object is designed to execute these methods automatically and hide the details from you.
- The following are the alarm clock object's private methods:
  - increment_current_second
  - increment_current_minute
  - increment_current_hour
  - sound_alarm

Classes

- The programmer determines the data attributes and methods that are necessary, then creates a *class*.
- A class is code that specifies the data attributes and methods of a particular type of object. Think of a class as a “blueprint” from which objects may be created.
- When we use the blueprint to build an actual house, we could say we are building an *instance* of the house described by the blueprint.

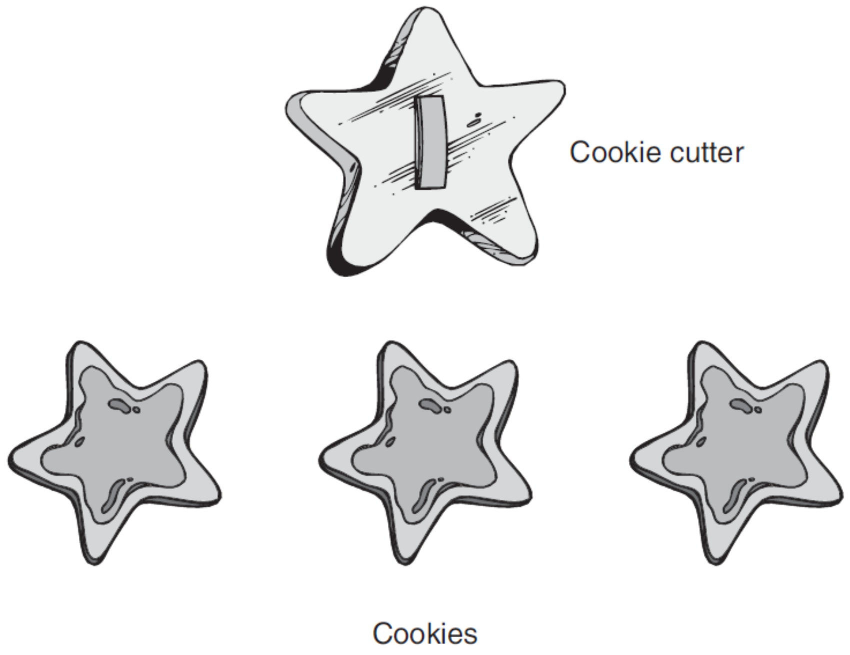
A blueprint and houses built from the blueprint

Blueprint that describes a house



Instances of the house described by the blueprint

- So, a class is a description of an object’s characteristics.
- When the program is running, it can use the class to create, in memory, as many objects of a specific type as needed.
- Each object that is created from a class is called an ***instance*** of the class.



Class Definitions

- To create a class, you write a *class definition*.
- A class definition is a set of statements that define a class's methods and data attributes.
- Suppose we are writing a program to simulate the tossing of a coin.
- In the program, we need to repeatedly toss the coin and each time determine whether it landed heads up or tails up.

```
class Coin:

    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.sideup = 'Heads'

    # The toss method generates a random number
    # in the range of 0 through 1. If the number
    # is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.

    def get_sideup(self):
        return self.sideup
```

The Coin class has three methods:

- ✓ The _ _init_ _ method
- ✓ The toss method
- ✓ The get_sideup method

- Most Python classes have a special method named _ _init_ _, which is automatically executed when an instance of the class is created in memory.
- The _ _init_ _ method is commonly known as an ***initializer method*** because it initializes the object's data attributes.
- Immediately after an object is created in memory, the _ _init_ _ method executes, and the **self** parameter is automatically assigned the object that was just created.

Aletheia University 資訊工程學系

```python
import random

# The Coin class simulates a coin that can
# be flipped.

class Coin:

    # The __init__ method initializes the
    # sideup data attribute with 'Heads'.

    def __init__(self):
        self.sideup = 'Heads'

    # The toss method generates a random number
    # in the range of 0 through 1. If the number
    # is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.

    def get_sideup(self):
        return self.sideup

# The main function.
def main():
    # Create an object from the Coin class.
    my_coin = Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am tossing the coin ...')
    my_coin.toss()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

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

Program Output

```
This side is up: Heads
I am tossing the coin ...
This side is up: Tails
```

Program Output

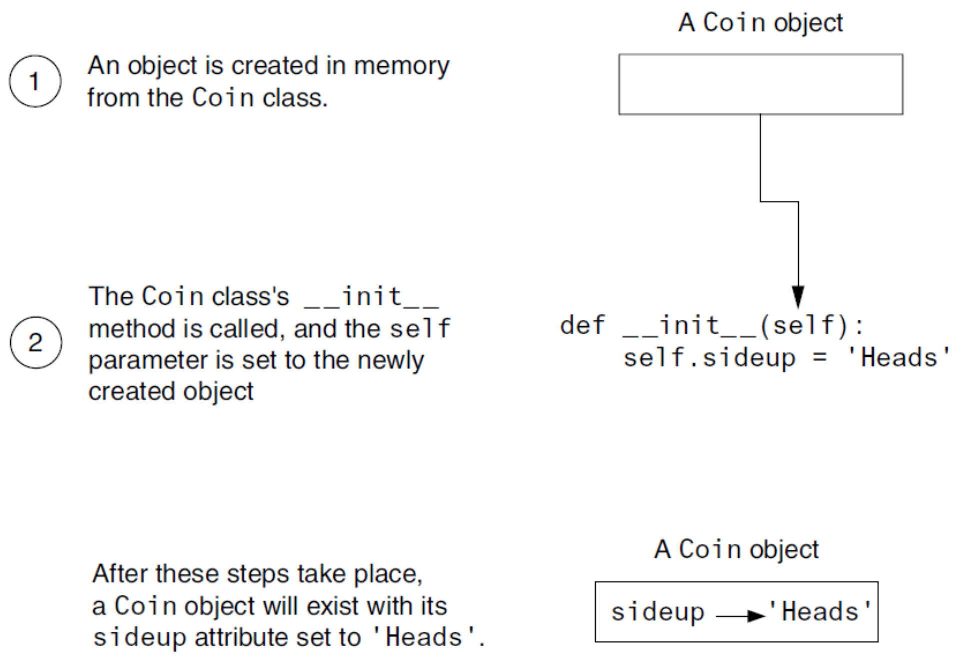
```
This side is up: Heads
I am tossing the coin ...
This side is up: Heads
```

Program Output

```
This side is up: Heads
I am tossing the coin ...
This side is up: Tails
```

- The expression Coin() that appears on the right side of the = operator causes two things to happen:
  1. An object is created in memory from the Coin class.
  2. The Coin class's _ _init_ _ method is executed, and the self parameter is automatically set to the object that was just created. As a result, that object's sideup attribute is assigned the string 'Heads'.

Actions Caused by the Coin() Expression



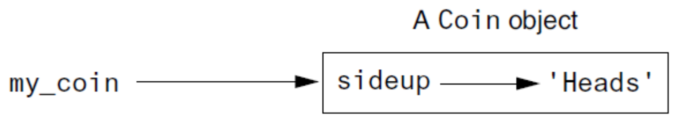
1. An object is created in memory from the Coin class.

2. The Coin class's `__init__` method is called, and the `self` parameter is set to the newly created object

```
def __init__(self):
    self.sideup = 'Heads'
```

After these steps take place, a Coin object will exist with its sideup attribute set to 'Heads'.

The my_coin Variable References a Coin Object



Hiding Attributes

- An object’s data attributes should be private, so that only the object’s methods can directly access them. This protects the object’s data attributes from accidental corruption.

```
def main():
    # Create an object from the Coin class.
    my_coin = Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am tossing the coin ...')
    my_coin.toss()

    # But now I'm going to cheat! I'm going to
    # directly change the value of the object's
    # sideup attribute to 'Heads'.
    my_coin.sideup = 'Heads'

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())
```

Aletheia University 資訊工程學系

- In Python, you can hide an attribute by starting its name with two underscore characters.
- If we change the name of the sideup attribute to _ _sideup, then code outside the Coin class will not be able to access it.

```
import random

# The Coin class simulates a coin that can
# be flipped.

class Coin:

    # The __init__ method initializes the
    # __sideup data attribute with ‘Heads’.

    def __init__(self):
    self.__sideup = 'Heads'

    # The toss method generates a random number
    # in the range of 0 through 1. If the number
    # is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.__sideup = 'Heads'
        else:
            self.__sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.

    def get_sideup(self):
        return self.__sideup

```

```
# The main function.
def main():
    # Create an object from the Coin class.
    my_coin = Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am going to toss the coin ten times:')
    for count in range(10):
        my_coin.toss()
        print(my_coin.get_sideup())
```

Storing Classes in Modules

- As programs use more classes, however, the need to organize those classes becomes greater.
- Programmers commonly organize their class definitions by storing them in modules.
- Then the modules can be imported into any programs that need to use the classes they contain.

coin.py

```python
import random

# The Coin class simulates a coin that can
# be flipped.

class Coin:

    # The __init__ method initializes the
    # __sideup data attribute with 'Heads'.

    def __init__(self):
        self.__sideup = 'Heads'

    # The toss method generates a random number
    # in the range of 0 through 1. If the number
    # is 0, then sideup is set to 'Heads'.
    # Otherwise, sideup is set to 'Tails'.

    def toss(self):
        if random.randint(0, 1) == 0:
            self.__sideup = 'Heads'
        else:
            self.__sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup.

    def get_sideup(self):
        return self.__sideup
```

Aletheia University 資訊工程學系

```python
# This program imports the coin module and
# creates an instance of the Coin class.

import coin

def main():
    # Create an object from the Coin class.
    my_coin = coin.Coin()

    # Display the side of the coin that is facing up.
    print('This side is up:', my_coin.get_sideup())

    # Toss the coin.
    print('I am going to toss the coin ten times:')
    for count in range(10):
        my_coin.toss()
        print(my_coin.get_sideup())

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

The BankAccount Class

```
class BankAccount:

    # The __init__ method accepts an argument for
    # the account's balance. It is assigned to
    # the __balance attribute.

    def __init__(self, bal):
        self.__balance = bal

    # The deposit method makes a deposit into the
    # account.

    def deposit(self, amount):
        self.__balance += amount

    # The withdraw method withdraws an amount
    # from the account.

    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    # The get_balance method returns the
    # account balance.

    def get_balance(self):
        return self.__balance
```

Aletheia University 資訊工程學系

```
import bankaccount

def main():
    # Get the starting balance.
    start_bal = float(input('Enter your starting balance: '))

    # Create a BankAccount object.
    savings = bankaccount.BankAccount(start_bal)

    # Deposit the user's paycheck.
    pay = float(input('How much were you paid this week? '))
    print('I will deposit that into your account.')
    savings.deposit(pay)

    # Display the balance.
    print('Your account balance is $',
          format(savings.get_balance(), ',.2f'),
          sep='')

    # Get the amount to withdraw.
    cash = float(input('How much would you like to withdraw? '))
    print('I will withdraw that from your account.')
    savings.withdraw(cash)

    # Display the balance.
    print('Your account balance is $',
          format(savings.get_balance(), ',.2f'),
          sep='')
```

The _ _str_ _ Method

- We need to display a message that indicates an object’s state.
- An object’s *state* is simply the values of the object’s attributes at any given moment.
- In Python, you give this method the special name _ _str_ _.

Aletheia University 資訊工程學系

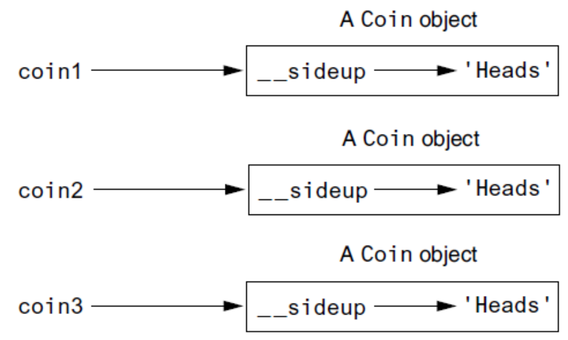
```python
class Rational:
    def __init__(self, n, d): # 物件建立之後所要建立的初始化動作
        self.numer = n
        self.denom = d

    def __str__(self): # 定義物件的字串描述
        return str(self.numer) + '/' + str(self.denom)

    def __add__(self, that): # 定義 + 運算
        return Rational(self.numer * that.denom + that.numer * self.denom, self.denom * that.denom)

    def __sub__(self, that): # 定義 - 運算
        return Rational(self.numer * that.denom - that.numer * self.denom, self.denom * that.denom)

    def __mul__(self, that): # 定義 * 運算
        return Rational(self.numer * that.numer, self.denom * that.denom)

    def __truediv__(self, that): # 定義 / 運算
        return Rational(self.numer * that.denom, self.denom * that.denom)

    def __eq__(self, that): # 定義 == 運算
        return self.numer * that.denom == that.numer * self.denom

x = Rational(1, 2)
y = Rational(2, 3)
z = Rational(2, 3)
print(x) # 1/2
print(y) # 2/3
print(x + y) # 7/6
print(x - y) # -1/6
print(x * y) # 2/6
print(x / y) # 3/6
print(x == y) # False
print(y == z) # True
```

Working with Instances

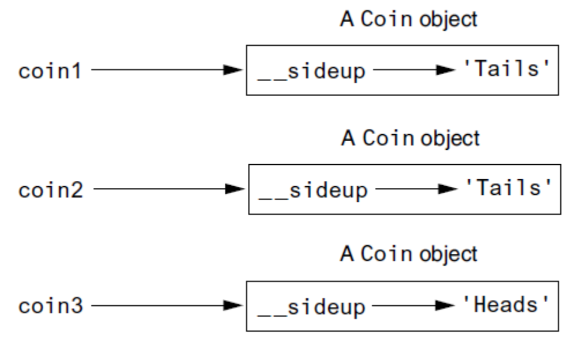
- Each instance of a class has its own set of data attributes.
- When a method uses the **self** parameter to create an attribute, the attribute belongs to the specific object that self references.
- We call these attributes ***instance attributes*** because they belong to a specific instance of the class.
- It is possible to create many instances of the same class in a program. Each instance will then have its own set of attributes.

```
def main():
    # Create three objects from the Coin class.
    coin1 = coin.Coin()
    coin2 = coin.Coin()
    coin3 = coin.Coin()

    # Display the side of each coin that is facing up.
    print('I have three coins with these sides up:')
    print(coin1.get_sideup())
    print(coin2.get_sideup())
    print(coin3.get_sideup())
    print()

    # Toss the coin.
    print('I am tossing all three coins ...')
    print()
    coin1.toss()
    coin2.toss()
    coin3.toss()

    # Display the side of each coin that is facing up.
    print('Now here are the sides that are up:')
    print(coin1.get_sideup())
    print(coin2.get_sideup())
    print(coin3.get_sideup())
    print()

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

The coin1, coin2, and coin3 variables reference three Coin objects



The objects after the toss method



Creating the CellPhone Class

Wireless Solutions, Inc. is a business that sells cell phones and wireless service. You are aprogrammer in the company's IT department, and your team is designing a program tomanage all of the cell phones that are in inventory. You have been asked to design a class that represents a cell phone. The data that should be kept as attributes in the class are as follows:

- The name of the phone's manufacturer will be assigned to the _ _manufact attribute.
- The phone's model number will be assigned to the _ _model attribute.
- The phone's retail price will be assigned to the _ _retail_price attribute.

The class will also have the following methods:

- An _ _init_ _ method that accepts arguments for the manufacturer, model number, and retail price.
- A set_manufact method that accepts an argument for the manufacturer. This method will allow us to change the value of the _ _manufact attribute after the object has been created, if necessary.
- A set_model method that accepts an argument for the model. This method will allow us to change the value of the _ _model attribute after the object has been created, if necessary.
- A set_retail_price method that accepts an argument for the retail price. This
- method will allow us to change the value of the _ _retail_price attribute after the object has been created, if necessary.
- A get_manufact method that returns the phone's manufacturer.
- A get_model method that returns the phone's model number.
- A get_retail_price method that returns the phone's retail price.

真理大學 Aletheia University 資訊工程學系

```
class CellPhone:

    # The __init__ method initializes the attributes.

    def __init__(self, manufact, model, price):
        self.__manufact = manufact
        self.__model = model
        self.__retail_price = price

    # The set_manufact method accepts an argument for
    # the phone's manufacturer.

    def set_manufact(self, manufact):
        self.__manufact = manufact
```

```
    # The set_model method accepts an argument for
    # the phone's model number.

    def set_model(self, model):
        self.__model = model

    # The set_retail_price method accepts an argument
    # for the phone's retail price.

    def set_retail_price(self, price):
        self.__retail_price = price

    # The get_manufact method returns the
    # phone's manufacturer.

    def get_manufact(self):
        return self.__manufact

    # The get_model method returns the
    # phone's model number.

    def get_model(self):
        return self.__model

    # The get_retail_price method returns the
    # phone's retail price.

    def get_retail_price(self):
        return self.__retail_price
```

```python
import cellphone

def main():
    # Get the phone data.
    man = input('Enter the manufacturer: ')


    mod = input('Enter the model number: ')
    retail = float(input('Enter the retail price: '))

    # Create an instance of the CellPhone class.
    phone = cellphone.CellPhone(man, mod, retail)

    # Display the data that was entered.
    print('Here is the data that you entered:')
    print('Manufacturer:', phone.get_manufact())
    print('Model Number:', phone.get_model())
    print('Retail Price: $', format(phone.get_retail_price(), ',.2f'), sep='')

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

Program Output (with input shown in bold)

```
Enter the manufacturer: Acme Electronics [Enter]
Enter the model number: M1000 [Enter]
Enter the retail price: 199.99 [Enter]
Here is the data that you entered:
Manufacturer: Acme Electronics
Model Number: M1000
Retail Price: $199.99
```

Aletheia University 資訊工程學系

Accessor and Mutator Methods

- As mentioned earlier, it is a common practice to make all of a class's data attributes private, and to provide public methods for accessing and changing those attributes.
- A method that returns a value from a class's attribute but does not change it is known as an *accessor method.*
- Accessor methods provide a safe way for code outside the class to retrieve the values of attributes, without exposing the attributes in a way that they could be changed by the code outside the method.
- In the CellPhone class section), the get_manufact, get_model, and get_retail_price methods are accessor methods.

- A method that stores a value in a data attribute or changes the value of a data attribute in some other way is known as a *mutator method*.
- Mutator methods can control the way that a class's data attributes are modified.
- When code outside the class needs to change the value of an object's data attribute, it typically calls a mutator and passes the new value as an argument.
- If necessary, the mutator can validate the value before it assigns it to the data attribute.
- The set_manufact, set_model, and set_retail_price methods are mutator methods.

Storing Objects in a List

- The CellPhone class you created in the previous *In the Spotlight* section will be used in a variety of programs. Many of these programs will store CellPhone objects in lists. To test the ability to store CellPhone objects in a list, you write the code in the previous program. This program gets the data for **five phones** from the user, creates five CellPhone objects holding that data, and stores those objects in a list. It then iterates over the list displaying the attributes of each object.

Program Output (with input shown in bold)

```
Enter data for five phones.

Phone number 1:
Enter the manufacturer: Acme Electronics [Enter]
Enter the model number: M1000 [Enter]
Enter the retail price: 199.99 [Enter]

Phone number 2:
Enter the manufacturer: Atlantic Communications [Enter]
Enter the model number: S2 [Enter]
Enter the retail price: 149.99 [Enter]

Phone number 3:
Enter the manufacturer: Wavelength Electronics [Enter]
Enter the model number: N477 [Enter]
Enter the retail price: 249.99 [Enter]

Phone number 4:
Enter the manufacturer: Edison Wireless [Enter]
Enter the model number: SLX88 [Enter]
Enter the retail price: 169.99 [Enter]

Phone number 5:
Enter the manufacturer: Sonic Systems [Enter]
Enter the model number: X99 [Enter]
Enter the retail price: 299.99 [Enter]

Here is the data you entered:
Acme Electronics
M1000
199.99
```

```
Atlantic Communications
S2
149.99

Wavelength Electronics
N477
249.99

Edison Wireless
SLX88
169.99

Sonic Systems
X99
299.99
```

Aletheia University 資訊工程學系

Passing Objects as Arguments

- When you are developing applications that work with objects, you often need to write functions and methods that accept objects as arguments.

```
def show_coin_status(coin_obj):
    print('This side of the coin is up:', coin_obj.get_sideup())
```

- The following code sample shows how we might create a Coin object, then pass it as an argument to the show_coin_status function:

```
my_coin = coin.Coin()
show_coin_status(my_coin)
```

```
import coin

# main function
def main():
    # Create a Coin object.
    my_coin = coin.Coin()

    # This will display 'Heads'.
    print(my_coin.get_sideup())

    # Pass the object to the flip function.
    flip(my_coin)

    # This might display 'Heads', or it might
    # display 'Tails'.
    print(my_coin.get_sideup())

# The flip function flips a coin.
def flip(coin_obj):
    coin_obj.toss()

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

Pickling Your Own Objects

- Recall from Chapter that the pickle module provides functions for serializing objects. Serializing an object means converting it to a stream of bytes that can be saved to a file for later retrieval. The pickle module's dump function serializes (pickles) an object and writes it to a file, and the load function retrieves an object from a file and deserializes (unpickles) it.

```
# This program pickles CellPhone objects.
import pickle
import cellphone

# Constant for the filename.
FILENAME = 'cellphones.dat'

def main():
    # Initialize a variable to control the loop.
    again = 'y'

    # Open a file.
    output_file = open(FILENAME, 'wb')

    # Get data from the user.
    while again.lower() == 'y':
        # Get cell phone data.
        man = input('Enter the manufacturer: ')
        mod = input('Enter the model number: ')
        retail = float(input('Enter the retail price: '))

        # Create a CellPhone object.
        phone = cellphone.CellPhone(man, mod, retail)

        # Pickle the object and write it to the file.
        pickle.dump(phone, output_file)

        # Get more cell phone data?
        again = input('Enter more phone data? (y/n): ')

    # Close the file.
    output_file.close()
    print('The data was written to', FILENAME)
```

An example that pickles three CellPhone objects and saves them to a file.

Program Output (with input shown in bold)

```
Enter the manufacturer: ACME Electronics [Enter]
Enter the model number: M1000 [Enter]
Enter the retail price: 199.99 [Enter]
Enter more phone data? (y/n): y [Enter]
Enter the manufacturer: Sonic Systems [Enter]
Enter the model number: X99 [Enter]
Enter the retail price: 299.99 [Enter]
Enter more phone data? (y/n): n [Enter]
The data was written to cellphones.dat
```

真理大學 Aletheia University 資訊工程學系

retrieves those objects from the file and unpickles them.

```
import pickle
import cellphone

# Constant for the filename.
FILENAME = 'cellphones.dat'

def main():
    end_of_file = False    # To indicate end of file

    # Open the file.
    input_file = open(FILENAME, 'rb')

    # Read to the end of the file.
    while not end_of_file:
        try:
            # Unpickle the next object.
            phone = pickle.load(input_file)

            # Display the cell phone data.
            display_data(phone)
        except EOFError:
            # Set the flag to indicate the end
            # of the file has been reached.
            end_of_file = True

    # Close the file.
    input_file.close()
```

```
# The display_data function displays the data
# from the CellPhone object passed as an argument.
def display_data(phone):
    print('Manufacturer:', phone.get_manufact())
    print('Model Number:', phone.get_model())
    print('Retail Price: $',
          format(phone.get_retail_price(), ',.2f'),
          sep='')
    print()

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

Program Output

```
Manufacturer: ACME Electronics
Model Number: M1000
Retail Price: $199.99

Manufacturer: Sonic Systems
Model Number: X99
Retail Price: $299.99
```

真理大學 Aletheia University 資訊工程學系

Storing Objects in a Dictionary

- Dictionaries are also useful for storing objects that you create from your own classes. Suppose you want to create a program that keeps contact information, such as names, phone numbers, and email addresses. You could start by writing a class such as the Contact class. An instance of the Contact class keeps the following data:
- A person's name is stored in the _ _name attribute.
- A person's phone number is stored in the _ _phone attribute.
- A person's email address is stored in the _ _email attribute.

- The class has the following methods:
  - ✓ An _ _init_ _ method that accepts arguments for a person's name, phone number, and email address
  - ✓ A set_name method that sets the _ _name attribute
  - ✓ A set_phone method that sets the _ _phone attribute
  - ✓ A set_email method that sets the _ _email attribute
  - ✓ A get_name method that returns the _ _name attribute
  - ✓ A get_phone method that returns the _ _phone attribute
  - ✓ A get_email method that returns the _ _email attribute
  - ✓ A _ _str_ _ method that returns the object's state as a string

Aletheia University 資訊工程學系

```
class Contact:
    # The __init__ method initializes the attributes.
    def __init__(self, name, phone, email):
        self.__name = name
        self.__phone = phone
        self.__email = email

    # The set_name method sets the name attribute.
    def set_name(self, name):
        self.__name = name

    # The set_phone method sets the phone attribute.
    def set_phone(self, phone):
        self.__phone = phone

    # The set_email method sets the email attribute.
    def set_email(self, email):
        self.__email = email

    # The get_name method returns the name attribute.
    def get_name(self):
        return self.__name

    # The get_phone method returns the phone attribute.
    def get_phone(self):
        return self.__phone

    # The get_email method returns the email attribute.
    def get_email(self):
        return self.__email

    # The __str__ method returns the object's state
    # as a string.
    def __str__(self):
        return "Name: " + self.__name + \
               "\nPhone: " + self.__phone + \
               "\nEmail: " + self.__email
```

- Next, you could write a program that keeps Contact objects in a dictionary. Each time the program creates a Contact object holding a specific person's data, that object would be stored as a value in the dictionary, using the person's name as the key. Then, any time you need to retrieve a specific person's data, you would use that person's name as a key to retrieve the Contact object from the dictionary.
- The program displays a menu that allows the user to perform any of the following operations:

✓ Look up a contact in the dictionary

✓ Add a new contact to the dictionary

✓ Change an existing contact in the dictionary

✓ Delete a contact from the dictionary

✓ Quit the program

Aletheia University 資訊工程學系

- Additionally, the program automatically pickles the dictionary and saves it to a file when the user quits the program. When the program starts, it automatically retrieves and unpickles the dictionary from the file. If the file does not exist, the program starts with an empty dictionary.
- The program is divided into eight functions: main, load_contacts, get_menu_choice, look_up, add, change, delete, and save_contacts. Rather than presenting the entire program at once, let's first examine the beginning part, which includes the import statements, global constants, and the main function.

```
Menu
---------------------------
1. Look up a contact
2. Add a new contact
3. Change an existing contact
4. Delete a contact
5. Quit the program

Enter your choice: 2 Enter
Name: Matt Goldstein Enter
Phone: 617-555-1234 Enter
Email: matt@fakecompany.com Enter
The entry has been added.

Menu
---------------------------
1. Look up a contact
2. Add a new contact
3. Change an existing contact
4. Delete a contact
5. Quit the program

Enter your choice: 2 Enter
Name: Jorge Ruiz Enter
Phone: 919-555-1212 Enter
Email: jorge@myschool.edu Enter
The entry has been added.

Menu
---------------------------
1. Look up a contact
2. Add a new contact
3. Change an existing contact
4. Delete a contact
5. Quit the program

Enter your choice: 5 Enter
```