EEE110 Computer Programming

Classes and Object-Oriented Programming

Dr Kasım Zor

May 2, 2023

Coin Class - Part 1/2

```
import random

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

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

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


Coin Class - Part 2/2

```
def main():
  my_coin = Coin()
  print('This side is up:', my_coin.get_sideup())
  print('I am tossing the coin ...')
  my_coin.toss()
  print('This side is up:', my_coin.get_sideup())

main()
```

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


Hiding Attributes - Part 1/3

```
def main():
  my_coin = Coin()
  print('This side is up:', my_coin.get_sideup())
  print('I am tossing the coin ...')
  my_coin.toss()
  my_coin.sideup = 'Heads' #for cheating
  print('This side is up:', my_coin.get_sideup())

main()
```

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

Hiding Attributes - Part 2/3 (coin.py)

```
import random

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

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

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


Hiding Attributes - Part 3/3

```
def main():
  my_coin = Coin()
  print('This side is up:', my_coin.get_sideup())
  print('I am tossing the coin three times')
  for count in range(3):
    my_coin.toss()
    print(my_coin.get_sideup())

main()
```

```
## This side is up: Heads
## I am tossing the coin three times
## Tails
## Tails
## Tails
```


Storing Classes in Modules

```
import coin #coin.py is in the same directory
def main():
  my_coin = coin.Coin()
  print('This side is up:', my_coin.get_sideup())
  print('I am tossing the coin three times')
  for count in range(3):
    my_coin.toss()
    print(my_coin.get_sideup())
main()
```

```
## This side is up: Heads
## I am tossing the coin three times
## Tails
## Tails
## Heads
```


The BankAccount Class (bankaccount.py)

```
class BankAccount:
  def __init__(self, bal):
    self.__balance = bal
  def deposit(self, amount):
    self.__balance += amount
  def withdraw(self, amount):
    if self.__balance >= amount:
      self.__balance -= amount
    else:
      print('Error: Insufficient funds')
  def get_balance(self):
    return self.__balance
```

Testing the Account - Part 1/2

```
import bankaccount #bankaccount.py in the same directory
def main():
  start_bal = 4400 #float(input('...'))
  savings = bankaccount.BankAccount(start_bal)
  pay = 600 #float(input('...'))
  print('I will deposit that into your account.')
  savings.deposit(pay)
  print('Your account balance is $',
        format(savings.get_balance(), ',.2f'), sep='')
  cash = 250 #float(input('...'))
  print('I will withdraw that from your account.')
  savings.withdraw(cash)
  print('Your account balance is $',
        format(savings.get_balance(), ',.2f'), sep='')
```


Testing the Account - Part 2/2

```
main()
```

```
## I will deposit that into your account.
## Your account balance is $5,000.00
## I will withdraw that from your account.
## Your account balance is $4,750.00
```


The _ _ str _ _ Method - Part 1/3 (bankaccount2.py)

```
class BankAccount:
  def __init__(self, bal): self.__balance = bal
  def deposit(self, amount): self.__balance += amount
  def withdraw(self, amount):
    if self.__balance >= amount: self.__balance -= amount
    else: print('Error: Insufficient funds')
  def get_balance(self): return self.__balance
  def __str__(self): #indicates object's state
                     #by returning a string
    return 'The balance is $' +\
           format(self.__balance, ',.2f')
```


The _ _ str _ _ Method - Part 2/3

```
import bankaccount2 #bankaccount2.py in the same directory

def main():
  start_bal = 4400 #float(input('...'))
  savings = bankaccount2.BankAccount(start_bal)
  pay = 600 #float(input('...'))
  print('I will deposit that into your account.')
  savings.deposit(pay)
  print(savings)
  cash = 250 #float(input('...'))
  print('I will withdraw that from your account.')
  savings.withdraw(cash)
  print(savings)
```

The __ __ str __ __ Method - Part 3/3

```
main()
```

```
## I will deposit that into your account.
## The balance is $5,000.00
## I will withdraw that from your account.
## The balance is $4,750.00
```


Question 1. Creating the CellPhone Class - Part 1/2 (15 minutes)

Wireless Solutions, Inc. is a business that sells cell phones and wireless service. You are a programmer in the company's IT department, and your team is designing a program to manage 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.

Question 1. Creating the CellPhone Class - Part 2/2 (15 minutes)

- 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.

Solution 1 - Part 1/2 (cellphone.py)

```
class CellPhone:
  def __init__(self, manufact, model, price):
    self.__manufact = manufact
    self.__model = model
    self.__retail_price = price
  def set_manufact(self, manufact): self.__manufact = manufact
  def set_model(self, model): self.__model = model
  def set_retail_price(self, price): self.__retail_price = pri
  def get_manufact(self):
    return self.__manufact
  def get_model(self):
    return self.__model
  def get_retail_price(self):
    return self.__retail_price
```

Solution 1 - Part 2/2

```
import cellphone #cellphone.py in the same directory
def main():
  man = 'Nokia' #input('...')
  mod = '3310' #input('...')
  retail = 100.0 # float(input('...'))
  phone = cellphone.CellPhone(man, mod, retail)
  print('You entered:\nManufacturer:', phone.get_manufact())
  print('Model Number:', phone.get_model())
  print('Retail Price: $', format(phone.get_retail_price(), '
main()
```

```
## You entered:
## Manufacturer: Nokia
## Model Number: 3310
## Retail Price: $100.00
```


Question 2. Storing Objects in a Dictionary - Part 1/2 (15 minutes)

Recall that dictionaries are objects that store elements as key-value pairs. Each element in a dictionary has a key and a value. If you want to retrieve a specific value from the dictionary, you do so by specifying its key. In Week 8, you saw examples that stored values such as strings, integers, floating-point numbers, lists, and tuples in dictionaries. Dictionaries are also useful for storing objects that you create from your own classes. Let's look at an example. Suppose you want to create a program that keeps contact information, such as names, phone numbers, and email addresses. 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.

Question 2. Storing Objects in a Dictionary - Part 2/2 (15 minutes)

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

Solution 2. (contact.py)

```
class Contact:
    def __init__(self, name, phone, email):
        self.__name = name
        self.__phone = phone
        self.__email = email
    def set_name(self, name): self.__name = name
    def set_phone(self, phone): self.__phone = phone
    def set_email(self, email): self.__email = email
    def get_name(self): return self.__name
    def get_phone(self): return self.__phone
    def get_email(self): return self.__email
    def __str__(self):
        return "Name: " + self.__name + "\nPhone: " +\
                self.__phone + "\nEmail: " + self.__email
```

Question 3. Contact - Part 1/2 (30 minutes)

Develop a Python 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

Question 3. Contact - Part 2/2 (30 minutes)

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,
- save_contacts.

Solution 3 - Part 1/7

```
import contact #contact.py in the same directory
import pickle

LOOK_UP = 1
ADD = 2
CHANGE = 3
DELETE = 4
QUIT = 5

#contacts.dat in the same directory
FILENAME = 'contacts.dat'
```


Solution 3 - Part 2/7

```
def main():
    mycontacts = load_contacts()
    choice = 0
    while choice != QUIT:
        choice = get_menu_choice()
        if choice == LOOK_UP:
            look_up(mycontacts)
        elif choice == ADD:
            add(mycontacts)
        elif choice == CHANGE:
            change(mycontacts)
        elif choice == DELETE:
            delete(mycontacts)
    save_contacts(mycontacts)
```

Solution 3 - Part 3/7

```
def load_contacts():
    try:
        input_file = open(FILENAME, 'rb')
        contact_dct = pickle.load(input_file)
        input_file.close()

    except IOError:
        contact_dct = {}

    return contact_dct
```


Solution 3 - Part 4/7

```
def get_menu_choice():
    print()
    print('Menu')
    print('---------------------------')
    print('1. Look up a contact')
    print('2. Add a new contact')
    print('3. Change an existing contact')
    print('4. Delete a contact')
    print('5. Quit the program')
    print()
    choice = int(input('Enter your choice: '))
    while choice < LOOK_UP or choice > QUIT:
        choice = int(input('Enter a valid choice: '))
    return choice
```


Solution 3 - Part 5/7

```
def look_up(mycontacts):
    name = input('Enter a name: ')
    print(mycontacts.get(name, 'That name is not found.'))

def add(mycontacts):
    name = input('Name: ')
    phone = input('Phone: ')
    email = input('Email: ')
    entry = contact.Contact(name, phone, email)
    if name not in mycontacts:
        mycontacts[name] = entry
        print('The entry has been added.')
    else:
        print('That name already exists.')
```


Solution 3 - Part 6/7

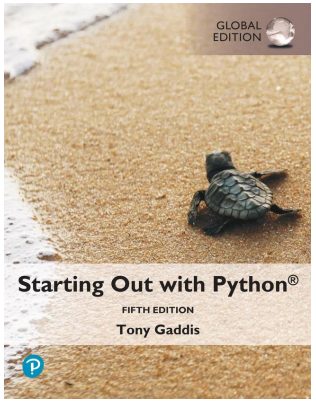
```
def change(mycontacts):
    name = input('Enter a name: ')
    if name in mycontacts:
        phone = input('Enter the new phone number: ')
        email = input('Enter the new email address: ')
        entry = contact.Contact(name, phone, email)
        mycontacts[name] = entry
        print('Information updated.')
    else:
        print('That name is not found.')
```

Solution 3 - Part 7/7

```
def delete(mycontacts):
    name = input('Enter a name: ')
    if name in mycontacts:
        del mycontacts[name]
        print('Entry deleted.')
    else:
        print('That name is not found.')

def save_contacts(mycontacts):
    output_file = open(FILENAME, 'wb')
    pickle.dump(mycontacts, output_file)
    output_file.close()

main()
```

Aforementioned contents are adapted from the book:

- 'Starting out with Python' written by Tony Gaddis.