Introduction to Programming using PYTHON

Session 6

N. D. Mendes
ndm@algos.inesc-id.pt

INESC-ID

October 23, 2006

Part I

Object-oriented Programming

Object-oriented programming versus Procedure-oriented programming

- In most applications, functionality requirements keep changing but the manipulated entities remain the same

Object-oriented programming versus Procedure-oriented programming

- In most applications, functionality requirements keep changing but the manipulated entities remain the same

$\Rightarrow$ Programming should be entity-oriented

Object-oriented programming versus Procedure-oriented programming

- In most applications, functionality requirements keep changing but the manipulated entities remain the same

$\Rightarrow$ Programming should be entity-oriented

e.g. Banking

- Operations: Calculating interests, currency conversions
- Entities: Accounts, Customers

Objects and Classes

- In object-oriented programming, program design is centered on the entities of the problem space: **Objects**

Object-oriented programming

Objects and Classes

- In object-oriented programming, program design is centered on the entities of the problem space: **Objects**
- A program consists of **Objects** interacting through messages called **Methods**

Object-oriented programming

Objects and Classes

- In object-oriented programming, program design is centered on the entities of the problem space: **Objects**
- A program consists of **Objects** interacting through messages called **Methods**
- Each **Object** can contain **Attributes**

Object-oriented programming

Objects and Classes

- In object-oriented programming, program design is centered on the entities of the problem space: **Objects**
- A program consists of **Objects** interacting through messages called **Methods**
- Each **Object** can contain **Attributes**
- The behaviour of an **Object** is defined by its **Class**

Object-oriented programming

Objects and Classes

- In object-oriented programming, program design is centered on the entities of the problem space: **Objects**
- A program consists of **Objects** interacting through messages called **Methods**
- Each **Object** can contain **Attributes**
- The behaviour of an **Object** is defined by its **Class**
- Every **Object** is an **Instance** of a **Class**

The `MobilePhone` Class

```
class MobilePhone:
   def __init__(self):
     self.__allCallsDuration = 0
     self.__allSentMessages = 0
     self.__outbox = []
```

The `MobilePhone` Class

```
class MobilePhone:
   def __init__(self):
     self.__allCallsDuration = 0
     self.__allSentMessages = 0
     self.__outbox = []


   def makeCall(self, duration):
     self.__allCallsDuration += duration
```

The `MobilePhone` Class

```
class MobilePhone:
   def __init__(self):
     self.__allCallsDuration = 0
     self.__allSentMessages = 0
     self.__outbox = []


   def makeCall(self, duration):
     self.__allCallsDuration += duration


   def sendSMS(self, message):
     self.__allSentMessages += 1
     self.__outbox.append(message)
```

The `MobilePhone` Class

```
class MobilePhone:
   def __init__(self):
     self.__allCallsDuration = 0
     self.__allSentMessages = 0
     self.__outbox = []


   def makeCall(self, duration):
     self.__allCallsDuration += duration


   def sendSMS(self, message):
     self.__allSentMessages += 1
     self.__outbox.append(message)


   def __str__(self):
     return """
       ALL CALLS = %s
       # SENT MESSAGES = %s
       OUTBOX:
       %s
       """ % (self.__allCallsDuration, self.__allSentMessages, '\n'.join(self.__outbox)
```

The `PrepaidMobilePhone` Class

```
class PrepaidMobilePhone(MobilePhone):
   INITIAL_CREDIT = 500
   TARIFF = 5


   def __init__(self):
     self.__credit = PrepaidMobilePhone.INITIAL_CREDIT
     MobilePhone.__init__(self)
```

The `PrepaidMobilePhone` Class

```
class PrepaidMobilePhone(MobilePhone):
   INITIAL_CREDIT = 500
   TARIFF = 5


   def __init__(self):
     self.__credit = PrepaidMobilePhone.INITIAL_CREDIT
     MobilePhone.__init__(self)


   def makeCall(self, duration):
     if (self.__credit >= duration * PrepaidMobilePhone.TARIFF):
       self.__credit -= duration * PrepaidMobilePhone.TARIFF
       MobilePhone.makeCall(self,duration)
     elif self.__credit > 0:
       actualDuration = float(self.__credit) / PrepaidMobilePhone.TARIFF
       self.__credit = 0
       MobilePhone.makeCall(self,actualDuration)
     else:
       raise NotEnoughCredit, "Not enough credit to make calls"
```

The `PrepaidMobilePhone` Class

```
class PrepaidMobilePhone(MobilePhone):
   INITIAL_CREDIT = 500
   TARIFF = 5


   def __init__(self):
     self.__credit = PrepaidMobilePhone.INITIAL_CREDIT
     MobilePhone.__init__(self)


   def makeCall(self, duration):
     if (self.__credit >= duration * PrepaidMobilePhone.TARIFF):
       self.__credit -= duration * PrepaidMobilePhone.TARIFF
       MobilePhone.makeCall(self,duration)
     elif self.__credit > 0:
       actualDuration = float(self.__credit) / PrepaidMobilePhone.TARIFF
       self.__credit = 0
       MobilePhone.makeCall(self,actualDuration)
     else:
       raise NotEnoughCredit, "Not enough credit to make calls"


   def sendSMS(self, message):
     if (self.__credit):
       MobilePhone.sendSMS(self, message)
     else:
       raise NotEnoughCredit, "Not enough credit to send messages"
```

The `PrepaidMobilePhone` Class

```
class PrepaidMobilePhone(MobilePhone):
   INITIAL_CREDIT = 500
   TARIFF = 5


   def __init__(self):
     self.__credit = PrepaidMobilePhone.INITIAL_CREDIT
     MobilePhone.__init__(self)


   def makeCall(self, duration):
     if (self.__credit >= duration * PrepaidMobilePhone.TARIFF):
       self.__credit -= duration * PrepaidMobilePhone.TARIFF
       MobilePhone.makeCall(self,duration)
     elif self.__credit > 0:
       actualDuration = float(self.__credit) / PrepaidMobilePhone.TARIFF
       self.__credit = 0
       MobilePhone.makeCall(self,actualDuration)
     else:
       raise NotEnoughCredit, "Not enough credit to make calls"


   def sendSMS(self, message):
     if (self.__credit):
       MobilePhone.sendSMS(self, message)
     else:
       raise NotEnoughCredit, "Not enough credit to send messages"

   def rechargeCredit(self, value):
     self.__credit += value
```

The `PrepaidMobilePhone` Class

```
class PrepaidMobilePhone(MobilePhone):
   INITIAL_CREDIT = 500
   TARIFF = 5


   def __init__(self):
     self.__credit = PrepaidMobilePhone.INITIAL_CREDIT
     MobilePhone.__init__(self)


   def makeCall(self, duration):
     if (self.__credit >= duration * PrepaidMobilePhone.TARIFF):
       self.__credit -= duration * PrepaidMobilePhone.TARIFF
       MobilePhone.makeCall(self,duration)
     elif self.__credit > 0:
       actualDuration = float(self.__credit) / PrepaidMobilePhone.TARIFF
       self.__credit = 0
       MobilePhone.makeCall(self,actualDuration)
     else:
       raise NotEnoughCredit, "Not enough credit to make calls"


   def sendSMS(self, message):
     if (self.__credit):
       MobilePhone.sendSMS(self, message)
     else:
       raise NotEnoughCredit, "Not enough credit to send messages"

   def __str__(self):
     return MobilePhone.__str__(self) + """ CREDIT = %s     """ % (self.__credit)
```

Instantiating Classes

```
from MobilePhone import *


gp = MobilePhone()
pp = PrepaidMobilePhone()


try:
   gp.makeCall(100)
   pp.makeCall(100)


   print isinstance(gp,PrepaidMobilePhone)
   print 'chargeCredit' in dir(gp)
   print 'chargeCredit' in dir(pp)
   pp.chargeCredit(1000)


   gp.sendSMS('hello world')
   pp.sendSMS('hello world')


except NotEnoughCredit, e:
   print "Credit Error:", e
except Exception, e:
   print "Unexpected Error:", e


print vars(gp)
print vars(pp)
```

Birds

Consider Birds, and their ability to tweet, fly, and drink from fountains. Consider also special kinds of Birds like Penguins and Swallows and their interactions with a single small Fountain with a limited amount of water.

Write a few classes trying to model the behaviour of this simple world and write a small program instantiating those classes.

For the next session

- From the manual
  - Read chapter 18
- A Series 2 will be given in the next and last session