Outline

1 Object-Oriented Design
   - unified modeling language
   - managing a library
   - modeling diagrams

2 Object-Oriented Programming in Python
   - class definitions and instantiations
   - data and functional attributes
   - classes for library manager

3 Summary + Assignments

MCS 260 Lecture 24
Introduction to Computer Science
Jan Verschelde, 7 March 2016
Object-Oriented Design

- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python

- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
Object-Oriented Programming (OOP) enables us to create our own high level data types, called abstract data types. Real-world entities (such as books, people) are represented in the software by objects and classes.

UML is a graphical language to model, design and construct object-oriented software. UML 2.1 defines 13 basic diagram types. Umbrello UML Modeller is a program for KDE on Knoppix.

Two types of modeling diagrams:
1. structural ones define the static architecture;
2. behavioral ones captures interactions and states.

Running example: library management system.
Object-Oriented Design

- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python

- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
Managing a Library

a case study

Goal: manage a library of books.

Two types of users: librarians and patrons.

Patrons when logged on may view the catalog, check out books, and return books.

After logging in, in addition to what is available to all, a librarian may

1. add and delete books;
2. add, search, and delete persons.

Still very simple management: only one person uses the program at any given time.
Object-Oriented Design

- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python

- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
the class Book
class diagram

An object of the class Book has three attributes:
identification number, title, availability.

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification number</td>
</tr>
<tr>
<td>title of book</td>
</tr>
<tr>
<td>availability status</td>
</tr>
</tbody>
</table>

| create new book       |
| show information      |
| check availability    |
| change availability   |

Four methods: __init__(), __str__(), check(), change().
the class Person

class diagram

An object of the class Person has three attributes: identification number, name, status.

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification number</td>
</tr>
<tr>
<td>name of person</td>
</tr>
<tr>
<td>status: librarian?</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>create new person</td>
</tr>
<tr>
<td>show information</td>
</tr>
<tr>
<td>check status</td>
</tr>
<tr>
<td>change status</td>
</tr>
</tbody>
</table>

Four methods: `__init__()`, `__str__()`, `check()`, `change()`.
The collection of books is an object of the class Catalog. Its one attribute collection is a list of books.

<table>
<thead>
<tr>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>list of books</td>
</tr>
<tr>
<td>add and delete</td>
</tr>
<tr>
<td>checkin and checkout</td>
</tr>
<tr>
<td>show the collection</td>
</tr>
<tr>
<td>search on key</td>
</tr>
</tbody>
</table>

In addition to `__init__()` and `__str__()` we have five methods: `add()`, `delete()`, `checkin()`, `checkout()`, and `search()`. The class Catalog imports from the class Book.
the class People

class diagram

An object of the class People has a list as first attribute. Its second attribute is who is currently logged on.

<table>
<thead>
<tr>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>list of persons</td>
</tr>
<tr>
<td>who is current</td>
</tr>
<tr>
<td>init with root</td>
</tr>
<tr>
<td>logon and logoff</td>
</tr>
<tr>
<td>who is logged on</td>
</tr>
<tr>
<td>add and delete</td>
</tr>
<tr>
<td>search on name</td>
</tr>
</tbody>
</table>

seven methods: `init()`, `logon()`, `logoff()`, `who()`, `add()`, `delete()` and `search()`.
Librarians and patrons differ in their use of the Catalog:

**Diagram:**

- **Librarian:**
  - Add
  - Delete
  - Show
  - Check in
  - Check out

- **Patron:**

**Catalog**
Librarians and patrons differ in their use of the People:

Librarian:
- Logon
- Add
- Delete
- Logout

Patron:

People
Design of a Library Manager

OOP follows bottom up design

Object-oriented design is typically bottom up, starting at the classes Book and Person.

The program `libman()` imports from Catalog and People. The class Catalog imports from Book and the class People imports from Person.
Object-Oriented Design

- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python

- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
class definitions and instantiations

The general syntax is

```python
class <class name>:
    <documentation string>
    <class data attributes>
    <methods>
```

As a general style rule, we will place a class definition in a separate file and treat it as a module.

Creating an instance of a class is called instantiation:

```python
<name of object> = <class name>()
```

Calling the class name as a function executes `__init__()` and returns an object.

We name this object by assigning it to a variable.
interactive usage of the class Book

If we store the definition of the class Book in the file `classbook.py`, then we work with objects as

```python
>>> from classbook import Book
>>> b1 = Book('one')
>>> print(b1)
 1: one is available
>>> b2 = Book('two')
>>> print(b2)
 2: two is available
>>> b3 = Book()
Give title : three
>>> print(b3)
 3: three is available
```
Object-Oriented Design
- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python
- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
data attributes: the class variables

Every object of our class Book has three data attributes: key, title, available.

class Book:
  "Objects of the class Book represent books."

Referencing an attribute goes like
<object>.<attribute name>

Session on previous slide continued:

>>> b1.key
1
>>> b1.title
'one'
class wide data attributes

To give each book a unique key, we count the number of books. Each time a new object is created, the counter is augmented by one, and the new object receives as key the new value of the counter.

```python
class Book(object):
    #
    # Objects of the class Book represent books.
    #
    count = [0]  # class wide attribute

    def __init__(self, *title):
        """constructor a a book"
        self.count[0] = self.count[0] + 1
        self.key = self.count[0]
```

The class wide attribute `count` is a list. Every object has a different list `count` as a data attribute, but the content of that list is the same.
functional attributes: the methods of a class

The constructor \texttt{\_\_init\_()} has an optional argument:

```python
def \_\_init\_(self, *title):
    "constructor a a a book"
    self.count[0] = self.count[0] + 1
    self.key = self.count[0]
    if len(title) > 0:
        self.title = title[0]
    else:
        self.title = input('Give title : ')
    self.available = True
```

The parameter \texttt{self} of \texttt{\_\_init\_()} is the instance itself.

We do not give an actual value for \texttt{self} as with other parameters. Instead:

```python
>>> from classbook import Book
>>> b = Book()
Give title :
```
Object-Oriented Design
- unified modeling language
- managing a library
- modeling diagrams

Object-Oriented Programming in Python
- class definitions and instantiations
- data and functional attributes
- classes for library manager

Summary + Assignments
class Book(object):
    """
    Objects of the class Book represent books.
    """
    def __init__(self, *title):
        """constructor a a book"
        """
    def __str__(self):
        """string representation of book"
        """
    def check(self):
        """returns the status of the book"
        """
    def change(self):
        """flips the status of the book"
        """
class Person(object):
    """
    An object of the class Person is either
    a librarian or a patron of the library.
    """

def __init__(self, **nps):
    """
    The arguments of the construct are name, PIN,
    and status, given in the dictionary nps.
    For each key not in nps, the user is prompted.
    """

def __str__(self):
    """string representation of a person"

    def check(self):
        """returns the status of the person"

    def change(self):
        """prompts for a new PIN and status"""
the method `__init__()` of Person

First part of the definition:

```python
def __init__(self, **nps):
    
    The arguments of the construct are name, PIN, and status, given in the dictionary `nps`. For each key not in `nps`, the user is prompted.
    
    if 'name' in nps:
        self.name = nps['name']
    else:
        self.name = input('Give your name : ')
```

The `name` is the key of the dictionary, used as

```python
>>> from classperson import Person
>>> p = Person(name='me', PIN=1234, status=True)
```
the method `__init__()` continued

Adding the value for `pin` and `librarian`:

```python
if 'PIN' in nps:
    self.pin = nps['PIN']
else:
    rawpin = input('Give your PIN : ')
    self.pin = int(rawpin)
if 'status' in nps:
    self.librarian = nps['status']
else:
    answer = input('Librarian ? (y/n) ')
    if answer == 'y':
        self.librarian = True
    else:
        self.librarian = False
```
interactive usage of the class Person

```python
>>> p = Person(name='me', PIN=1234, status=True)
>>> str(p)
'me with PIN 1234 is a librarian'
>>> p.name
'me'
>>> p.pin
1234
>>> p.librarian
True
```
the class Catalog encapsulates a list

from classbook import Book

class Catalog(object):
    
    """
    The class Catalog holds the book collection.
    """
    def __init__(self):
        """sets the data attribute"
        self.collection = []

    def __str__(self):
        """returns the string representation"
        result = ""
        for book in self.collection:
            result += str(book) + ' \
        return result

    def add(self, book):
        """adds the book to the catalog"
        self.collection.append(book)
interactive usage of the Catalog

```python
>>> from classcatalog import Catalog
>>> from classbook import Book
>>> b = Book()
Give title : one
>>> c = Catalog()
>>> c.add(b)
>>> print(c)
  1: one is available

>>> print(c.search_on_key(1))
  1: one is available

We add a method to search on the key:

```
def search_on_key(self, key):
    """
    Returns the book with the key if it is in
    the collection, else None is returned.
    """
    for book in self.collection:
        if book.key == key:
            return book
    return None
def checkout(self, key):
    "Checks out the book with key."
book = self.search_on_key(key)
if book == None:
    print('no book with key = ', key)
else:
    if not book.check():
        print(book)
    else:
        book.change()
def checkin(self, key):
    "Checks in the book with key."
    book = self.search_on_key(key)
    if book == None:
        print('no book with key = ', key)
    elif isinstance(book, Book):
        if not book.check():
            book.change()
def delete(self, key):
    "Deletes the book with key."
    for k in range(0, len(self.collection)):
        book = self.collection[k]
        if book.key == key:
            popped = self.collection.pop(k)
            print(popped)
            print('has been deleted')
            break
the class People

Data attributes: a list and the name of the current user.

from classperson import Person

class People(object):
    """
    The class People collects information of all librarians and patrons of the library.
    """
    def __init__(self):
        """Creates a root user.""
        root = Person(name='root', PIN=0, status=True)
        self.whoswho = [root]
        self.current = ''

The __init__ constructor method is invoked at the time of instantiation: it initializes the object.
the class People continued

def search(self, name):
    """
    Returns None if name not in self.whoswho, else the person object is returned.
    """

    for person in self.whoswho:
        if person.name == name:
            return person
    return None

def logon(self):
    """
    Prompts for name and PIN, and returns
    -1 if access is not granted;
    0 if the user is a patron;
    +1 if the user is a librarian.
    """
the method `logon()` defined

```python
name = input('Give your name : ')
member = self.search(name)
if member == None:
    print('you are not a member')
    return -1
else:
    rawpin = input('Give your PIN : ')
    intpin = int(rawpin)
    if intpin == member.pin:
        print('welcome ' + member.name)
        self.current = member
        if member.librarian:
            return +1
        else:
            return 0
    else:
        print('wrong PIN')
        return -1
```
def who(self):
    "Shows who is currently logged in."
    if self.current == '':
        print('nobody is logged in')
    else:
        print(self.current.name + ' is logged in')

def logoff(self):
    "The current user is logged off."
    if self.current != '':
        print('bye bye, ' + self.current.name)
        self.current = ''
def add(self, *person):
    "Adds a new person to the collection."
    if len(person) > 0:
        toadd = person[0]
    else:
        toadd = Person()
    self.whoswho.append(toadd)

def delete(self):
    "Prompts for a name and then deletes."
    name = input('Give name : ')
    for k in range(0, len(self.whoswho)):
        person = self.whoswho[k]
        if person.name == name:
            popped = self.whoswho.pop(k)
            print(popped, 'has been deleted')
            break
main program: libclassman.py

import Catalog
import People

def show_menu(who):
    
    Shows the menu to the user
    and returns the choice made.

    def act(pep, cat, chc, who):
        "Performs the requested action."

    def main():
        "Main library management program."
        cat = Catalog()
        people = People()
        who = -1
        print('Welcome to our library!')
        while True:
            choice = show_menu(who)
            if choice == 9:
                break
            who = act(people, cat, choice, who)
the function `show_menu`

def show_menu(who):
    """
    Shows the menu to the user
    and returns the choice made.
    """
    if who == -1:  # no one logged on
        print('Please log on')
        return 0
    else:  # who == 0 is patron
        print('choose from the menu :')
        print('  1. log off')
        print('  2. show the collection')
        print('  3. check out a book')
        print('  4. return a book')
        if who == +1:  # librarian
            print('  5. add a new book')
            print('  6. delete a book')
            print('  7. add a new user')
            print('  8. delete a user')
            print('  9. shut down')
        ansraw = input('Make your choice : ')
        return int(ansraw)
def act(pep, cat, chc, who):
    "Performs the requested action."
    result = who
    if chc == 0:
        result = pep.logon()
    elif chc == 1:
        pep.logoff()
    result = -1
    elif chc == 2:
        cat.show()
    elif chc == 3:
        cat.checkout()
    elif chc == 4:
        cat.checkin()
    elif chc == 5:
        cat.add()
    elif chc == 6:
        cat.delete()
    elif chc == 7:
        pep.add()
    elif chc == 8:
        pep.delete()
    return result
Summary + Assignments

We started chapter 10 in *Python Programming*, see §6.5 in *Computer Science, an overview* for UML, see online tutorials, at [www.uml.org](http://www.uml.org).

Assignments:

1. **Make a class** `Counter` **which initializes to zero.** The method `add` increments the counter by one. The string representation returns the value of the counter, that is: the value of the data attribute stored by the object instantiated from the class `Counter`.

2. **Design a class** `Rational` **to compute with rational numbers.** Ensure that a rational number is always normalized: numerator and denominator have 1 as their only common divisor.

3. **Write Python code for the class** `Rational`.

4. **Describe how the design of our library manager would change if files would be used for the catalog and people.** Which functions would change?