

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

- 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 Vershelde, 8 March 2010

# object-oriented design

## OOP in Python

### 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

- 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

# Object-Oriented Design

## UML: Unified Modeling Language

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

## UML: Unified Modeling Language

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

## UML: Unified Modeling Language

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

## UML: Unified Modeling Language

### 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

## UML: Unified Modeling Language

### 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

## OOP in Python

### 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

## 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

# 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.

# 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.

# 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.

# 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.

# 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.

# 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

## OOP in Python

### 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

## 1 Object-Oriented Design

unified modeling language  
managing a library  
**modeling diagrams**

## 2 Object-Oriented Programming in Python

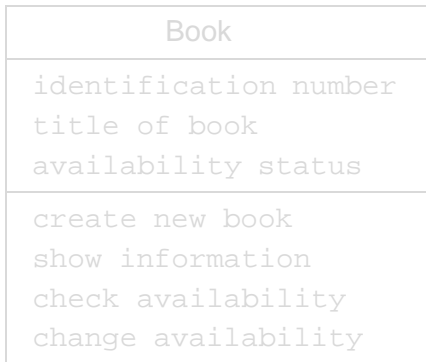
class definitions and instantiations  
data and functional attributes  
classes for library manager

## 3 Summary + Assignments

# The Class Book

## class diagram

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

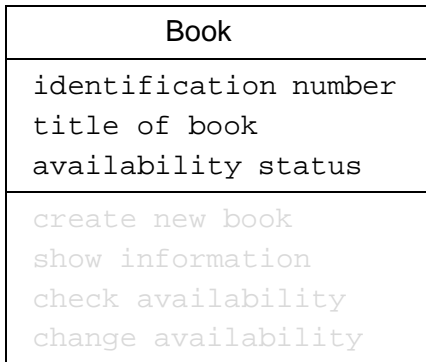


4 methods: `create()`, `show()`, `check()`, `change()`.

# The Class Book

## class diagram

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

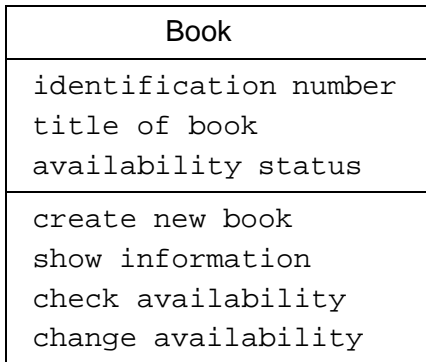


4 methods: `create()`, `show()`, `check()`, `change()`.

# The Class Book

class diagram

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

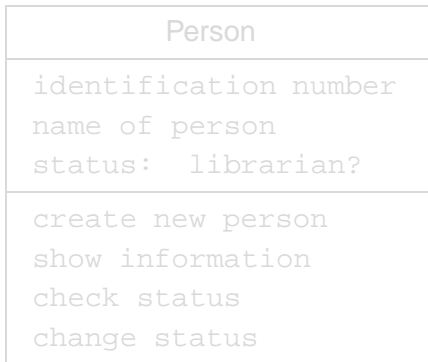


4 methods: `create()`, `show()`, `check()`, `change()`.

# The Class Person

class diagram

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

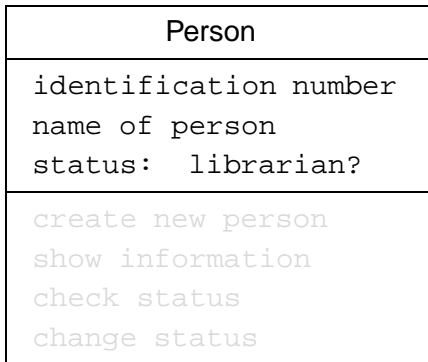


4 methods: `create()`, `show()`, `check()`, `change()`.

# The Class Person

class diagram

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



4 methods: `create()`, `show()`, `check()`, `change()`.

# The Class Person

class diagram

## Object-Oriented Design

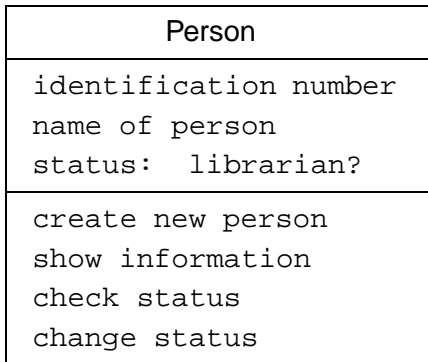
unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

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

## Summary + Assignments

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



4 methods: create(), show(), check(), change().

# The Class Catalog

class diagram

The collection of books is an object of the class `Catalog`.  
Its one attribute `collection` is a list of books.



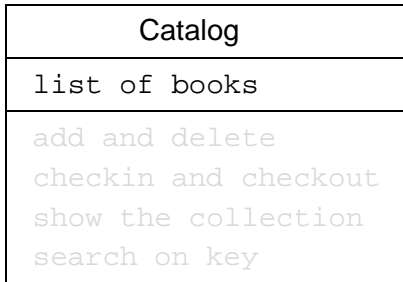
six methods: `add()`, `delete()`, `checkin()`,  
`checkout()`, `show()`, and `search()`.

The class `Catalog` imports from the class `Book`.

# The Class Catalog

class diagram

The collection of books is an object of the class `Catalog`. Its one attribute `collection` is a list of books.



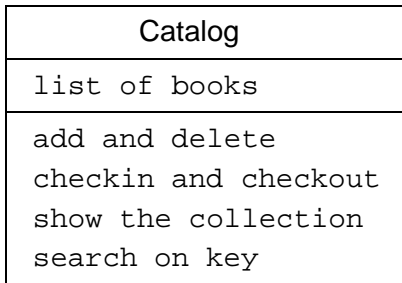
six methods: `add()`, `delete()`, `checkin()`, `checkout()`, `show()`, and `search()`.

The class `Catalog` imports from the class `Book`.

# The Class Catalog

class diagram

The collection of books is an object of the class `Catalog`.  
Its one attribute `collection` is a list of books.



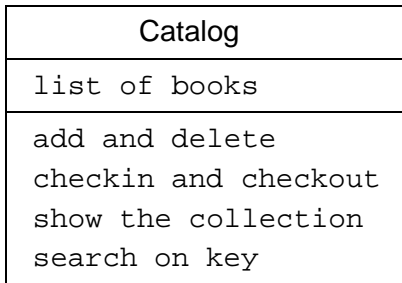
**six methods:** `add()`, `delete()`, `checkin()`,  
`checkout()`, `show()`, **and** `search()`.

The class `Catalog` imports from the class `Book`.

# The Class Catalog

class diagram

The collection of books is an object of the class `Catalog`.  
Its one attribute `collection` is a list of books.



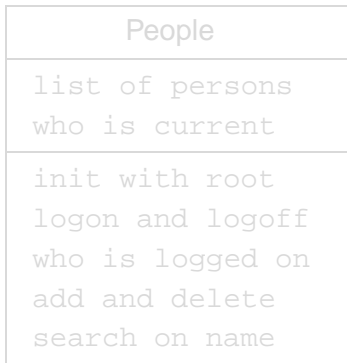
**six methods:** `add()`, `delete()`, `checkin()`,  
`checkout()`, `show()`, and `search()`.

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.

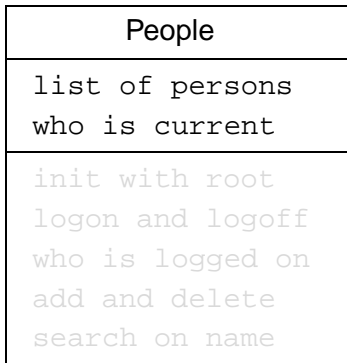


seven methods: `init()`, `logon()`, `logoff()`, `who()`,  
`add()`, `delete()` and `search()`.

# 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.

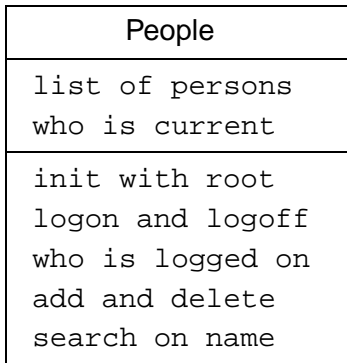


seven methods: `init()`, `logon()`, `logoff()`, `who()`,  
`add()`, `delete()` and `search()`.

# 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.



seven methods: `init()`, `logon()`, `logoff()`, `who()`,  
`add()`, `delete()` and `search()`.

8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

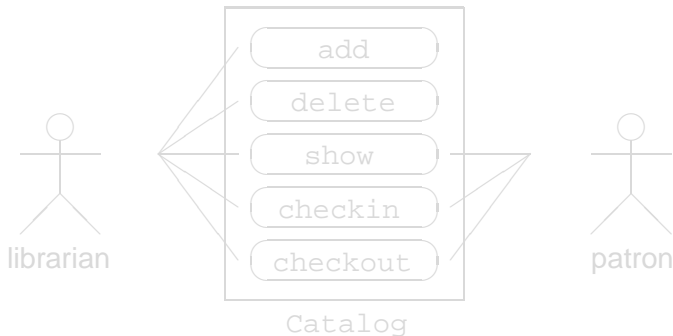
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for Catalog

a behavior modeling diagram

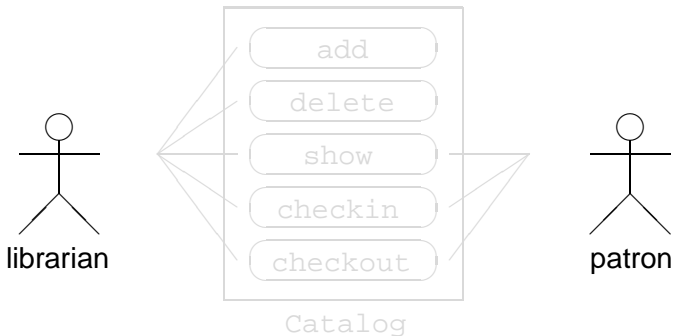
Librarians and patrons differ in their use of the Catalog:



# Use Case Diagram for Catalog

a behavior modeling diagram

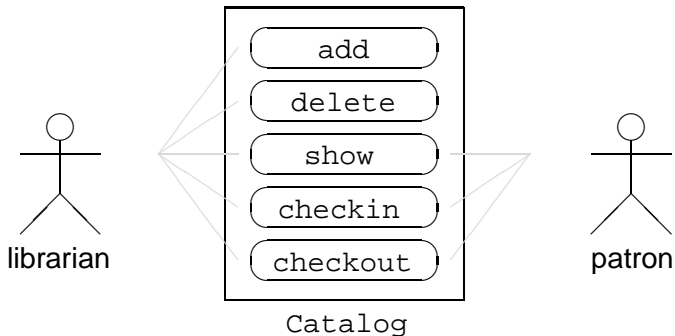
Librarians and patrons differ in their use of the Catalog:



# Use Case Diagram for Catalog

a behavior modeling diagram

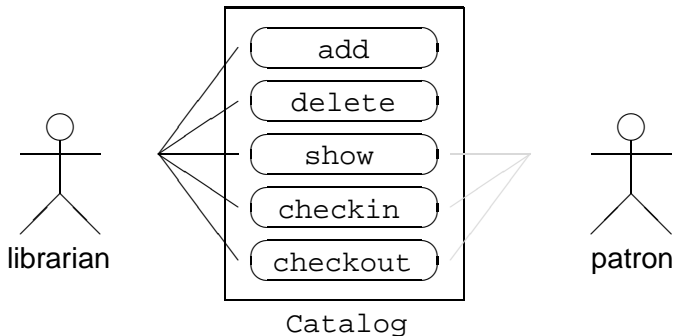
Librarians and patrons differ in their use of the Catalog:



# Use Case Diagram for Catalog

a behavior modeling diagram

Librarians and patrons differ in their use of the Catalog:



8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

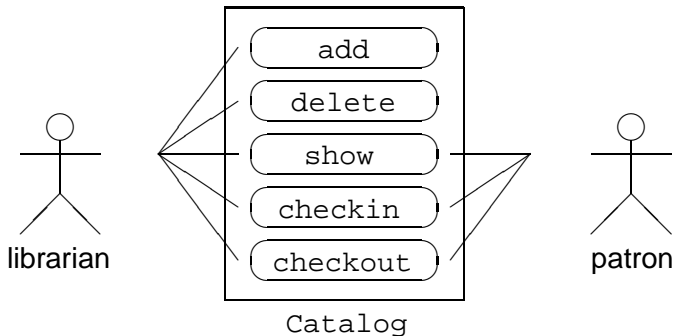
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for Catalog

a behavior modeling diagram

Librarians and patrons differ in their use of the Catalog:



8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

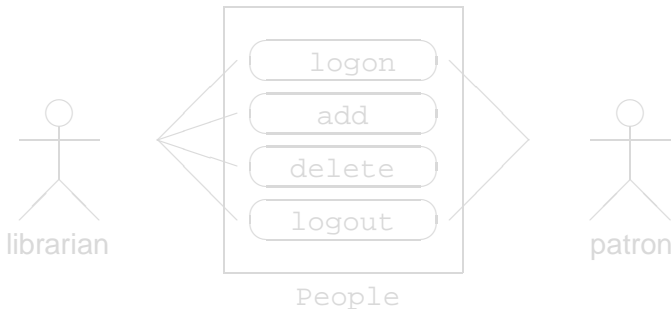
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for People

a behavior modeling diagram

Librarians and patrons differ in their use of the People:



8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

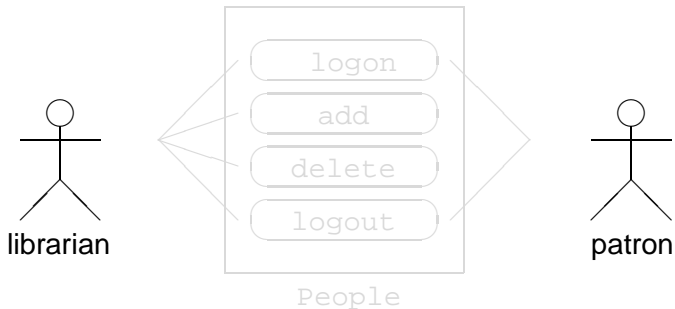
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for People

a behavior modeling diagram

Librarians and patrons differ in their use of the People:



8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

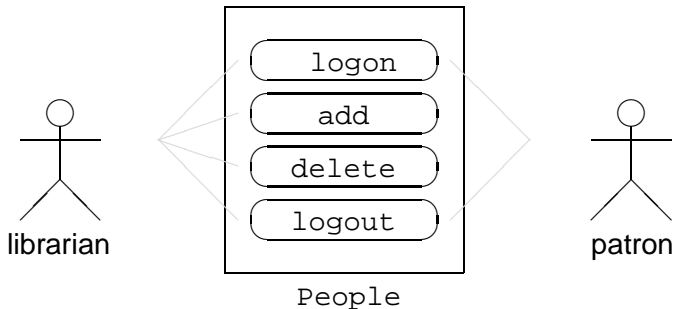
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for People

a behavior modeling diagram

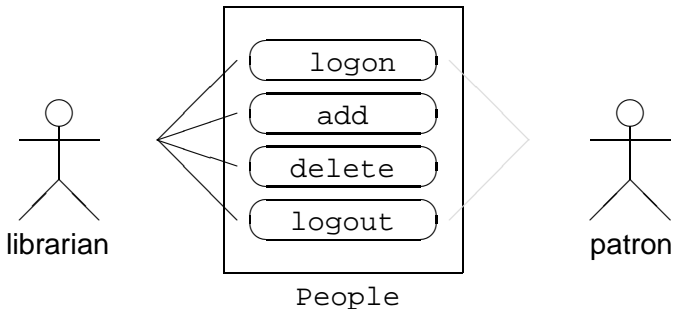
Librarians and patrons differ in their use of the People:



# Use Case Diagram for People

a behavior modeling diagram

Librarians and patrons differ in their use of the People:



8 Mar 2010

## Object-Oriented Design

unified modeling language  
managing a library  
modeling diagrams

## Object-Oriented Programming in Python

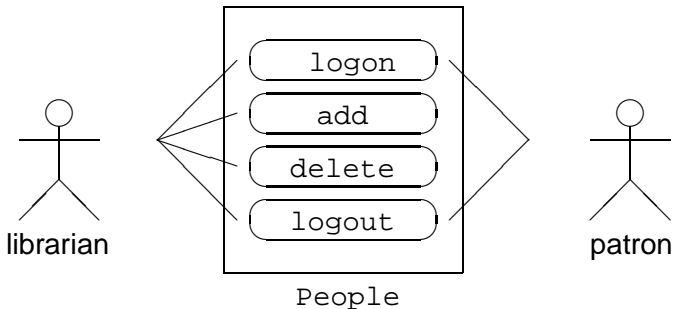
class definitions and instantiations  
data and functional attributes  
classes for library manager

## Summary + Assignments

# Use Case Diagram for People

a behavior modeling diagram

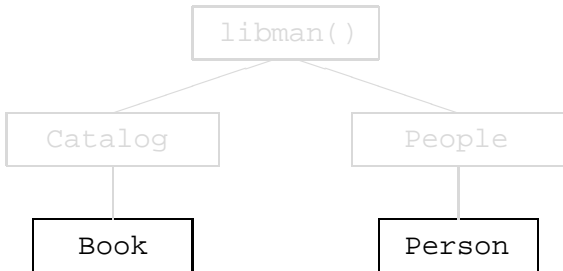
Librarians and patrons differ in their use of the People:



# Design of a Library Manager

OOP follows bottom up design

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

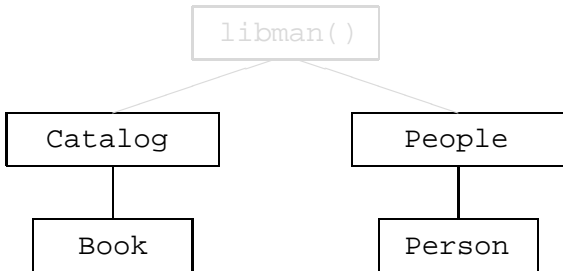


The program `libman()` imports from `Catalog` and `People`.  
The class `Catalog` imports from `Book`  
and the class `People` imports from `Person`.

# Design of a Library Manager

OOP follows bottom up design

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

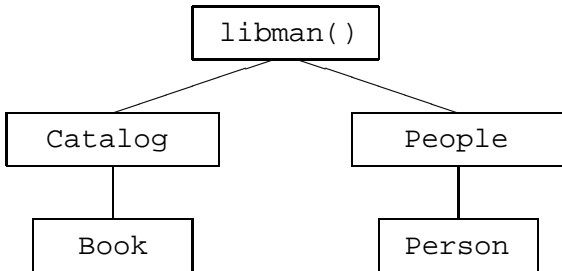


The program `libman()` imports from `Catalog` and `People`.  
The class `Catalog` imports from `Book`  
and the class `People` imports from `Person`.

# Design of a Library Manager

OOP follows bottom up design

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

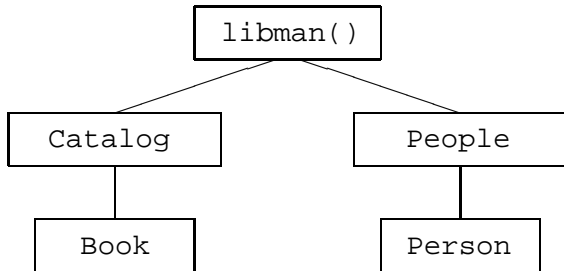


The program `libman()` imports from `Catalog` and `People`.  
The class `Catalog` imports from `Book`  
and the class `People` imports from `Person`.

# 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

## OOP in Python

### 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

- 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

# Class Definitions and Instantiations

object-oriented programming in Python

The general syntax is

```
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:

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

Calling the class name as a function gives an empty object. We name this object by assigning it to a variable.

# Class Definitions and Instantiations

object-oriented programming in Python

The general syntax is

```
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:

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

Calling the class name as a function gives an empty object.  
We name this object by assigning it to a variable.

# Class Definitions and Instantiations

object-oriented programming in Python

The general syntax is

```
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:

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

Calling the class name as a function gives an empty object. We name this object by assigning it to a variable.

8 Mar 2010

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 design

## OO in Python

- 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

# 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 >
```

For example (using file `classbook.py`):

```
>>> from classbook import *
>>> b = Book()
>>> b.key = 3
>>> b.key
3
```

# 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 >
```

For example (using file `classbook.py`):

```
>>> from classbook import *
>>> b = Book()
>>> b.key = 3
>>> b.key
3
```

# 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 >
```

For example (using file `classbook.py`):

```
>>> from classbook import *
>>> b = Book()
>>> b.key = 3
>>> b.key
3
```

# Functional Attributes

the methods of a class

Methods are the functions defined in a class.

Continuing with the file `classbook.py`, we have:

```
def create(self):  
    "Prompts the user for number and title."  
    self.key = input('Give number : ')  
    self.title = raw_input('Give title : ')  
    self.available = True  
    return self
```

The parameter `self` of `create` is the instance itself.

We do not give an actual value for `self` as with other parameters. Instead:

```
>>> from classbook import *  
>>> b = Book()  
>>> b.create()  
Give number :
```

# Functional Attributes

the methods of a class

Methods are the functions defined in a class.  
Continuing with the file `classbook.py`, we have:

```
def create(self):  
    "Prompts the user for number and title."  
    self.key = input('Give number : ')  
    self.title = raw_input('Give title : ')  
    self.available = True  
    return self
```

The parameter `self` of `create` is the instance itself.

We do not give an actual value for `self` as with other parameters. Instead:

```
>>> from classbook import *  
>>> b = Book()  
>>> b.create()  
Give number :
```

# Functional Attributes

the methods of a class

Methods are the functions defined in a class.

Continuing with the file `classbook.py`, we have:

```
def create(self):  
    "Prompts the user for number and title."  
    self.key = input('Give number : ')  
    self.title = raw_input('Give title : ')  
    self.available = True  
    return self
```

The parameter `self` of `create` is the instance itself.

We do not give an actual value for `self` as with other parameters. Instead:

```
>>> from classbook import *  
>>> b = Book()  
>>> b.create()  
Give number :
```

# Functional Attributes

the methods of a class

Methods are the functions defined in a class.

Continuing with the file `classbook.py`, we have:

```
def create(self):  
    "Prompts the user for number and title."  
    self.key = input('Give number : ')  
    self.title = raw_input('Give title : ')  
    self.available = True  
    return self
```

The parameter `self` of `create` is the instance itself.

We do not give an actual value for `self` as with other parameters. Instead:

```
>>> from classbook import *  
>>> b = Book()  
>>> b.create()  
Give number :
```

# object-oriented design

## OO in Python

### 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

- 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

8 Mar 2010

Object-  
Oriented  
Designunified modeling  
language  
managing a library  
modeling diagramsObject-  
Oriented  
Programming  
in Pythonclass definitions and  
instantiations  
data and functional  
attributes  
**classes for library  
manager**Summary +  
Assignments

# The Class Book

the file classbook.py

```
class Book:
    "Objects of the class Book represent books."
    def create(self):
        "Prompts the user for number and title."
    def show(self):
        "Shows content of the book."
    def check(self):
        "Returns the status of the book."
    def change(self):
        "Flips the status of the book."
```

# The Class Book

the file classbook.py

```
class Book:
    "Objects of the class Book represent books."
    def create(self):
        "Prompts the user for number and title."
    def show(self):
        "Shows content of the book."
    def check(self):
        "Returns the status of the book."
    def change(self):
        "Flips the status of the book."
```

# The Class Book

the file classbook.py

```
class Book:
    "Objects of the class Book represent books."
    def create(self):
        "Prompts the user for number and title."
    def show(self):
        "Shows content of the book."
    def check(self):
        "Returns the status of the book."
    def change(self):
        "Flips the status of the book."
```

# The Class Book

the file classbook.py

```
class Book:
    "Objects of the class Book represent books."
    def create(self):
        "Prompts the user for number and title."
    def show(self):
        "Shows content of the book."
    def check(self):
        "Returns the status of the book."
    def change(self):
        "Flips the status of the book."
```

# The Class Book

the file classbook.py

```
class Book:
    "Objects of the class Book represent books."
    def create(self):
        "Prompts the user for number and title."
    def show(self):
        "Shows content of the book."
    def check(self):
        "Returns the status of the book."
    def change(self):
        "Flips the status of the book."
```

# The Class Person

the file classperson.py

```
class Person:
    """
    An object of the class Person is either
    a librarian or a patron of the Library.
    """

    def create(self, **nps):
        "Prompts for data if not in nps."

    def show(self):
        "Shows info about the person."

    def check(self):
        "Returns the status of the person."

    def change(self):
        "Prompts for a new PIN and status."
```

# The Class Person

the file classperson.py

```
class Person:
    """
    An object of the class Person is either
    a librarian or a patron of the Library.
    """
    def create(self, **nps):
        "Prompts for data if not in nps."

    def show(self):
        "Shows info about the person."

    def check(self):
        "Returns the status of the person."

    def change(self):
        "Prompts for a new PIN and status."
```

# The Class Person

the file classperson.py

```
class Person:
    """
    An object of the class Person is either
    a librarian or a patron of the Library.
    """
    def create(self, **nps):
        "Prompts for data if not in nps."
    def show(self):
        "Shows info about the person."
    def check(self):
        "Returns the status of the person."
    def change(self):
        "Prompts for a new PIN and status."
```

# The Class Person

the file classperson.py

```
class Person:
    """
    An object of the class Person is either
    a librarian or a patron of the Library.
    """
    def create(self, **nps):
        "Prompts for data if not in nps."
    def show(self):
        "Shows info about the person."
    def check(self):
        "Returns the status of the person."
    def change(self):
        "Prompts for a new PIN and status."
```

# The Class Person

the file classperson.py

```
class Person:
    """
    An object of the class Person is either
    a librarian or a patron of the Library.
    """
    def create(self, **nps):
        "Prompts for data if not in nps."
    def show(self):
        "Shows info about the person."
    def check(self):
        "Returns the status of the person."
    def change(self):
        "Prompts for a new PIN and status."
```

# The Function create of 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

```
def create(self, **nps):
    "Prompts for data if not in nps."
    if nps.has_key('name'):
        self.name = nps['name']
    else:
        self.name = raw_input('Give your name : ')
    if nps.has_key('PIN'):
        self.PIN = nps['PIN']
    else:
        self.PIN = input('Give your PIN : ')
    if nps.has_key('status'):
        self.librarian = nps['status']
    else:
        answer = raw_input('Librarian ? (y/n) ')
        if answer == 'y':
            self.librarian = True
        else:
            self.librarian = False
    return self
```

# The Function create of 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

```
def create(self, **nps):
    "Prompts for data if not in nps."
    if nps.has_key('name'):
        self.name = nps['name']
    else:
        self.name = raw_input('Give your name : ')
    if nps.has_key('PIN'):
        self.PIN = nps['PIN']
    else:
        self.PIN = input('Give your PIN : ')
    if nps.has_key('status'):
        self.librarian = nps['status']
    else:
        answer = raw_input('Librarian ? (y/n) ')
        if answer == 'y':
            self.librarian = True
        else:
            self.librarian = False
    return self
```

# The Function create of 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

```
def create(self, **nps):
    "Prompts for data if not in nps."
    if nps.has_key('name'):
        self.name = nps['name']
    else:
        self.name = raw_input('Give your name : ')
    if nps.has_key('PIN'):
        self.PIN = nps['PIN']
    else:
        self.PIN = input('Give your PIN : ')
    if nps.has_key('status'):
        self.librarian = nps['status']
    else:
        answer = raw_input('Librarian ? (y/n) ')
        if answer == 'y':
            self.librarian = True
        else:
            self.librarian = False
    return self
```

# The Function create of 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

```
def create(self, **nps):
    "Prompts for data if not in nps."
    if nps.has_key('name'):
        self.name = nps['name']
    else:
        self.name = raw_input('Give your name : ')
    if nps.has_key('PIN'):
        self.PIN = nps['PIN']
    else:
        self.PIN = input('Give your PIN : ')
    if nps.has_key('status'):
        self.librarian = nps['status']
    else:
        answer = raw_input('Librarian ? (y/n) ')
        if answer == 'y':
            self.librarian = True
        else:
            self.librarian = False
    return self
```

# The Class Catalog

the file `classcatalog.py`

The collection of books is represented as a list,  
as the data attribute of the class `Catalog`.

```
class Catalog:
    """
    The class Catalog imports the class Book.
    It represents the library's book collection.
    """
    collection = []

    def add(self):
        "Prompts the user for number and title."
        import classbook
        b = classbook.Book()
        b.create()
        self.collection.append(b)
```

The class `Catalog` imports the class `Book`.

# The Class Catalog

the file `classcatalog.py`

The collection of books is represented as a list,  
as the data attribute of the class `Catalog`.

```
class Catalog:
    """
    The class Catalog imports the class Book.
    It represents the library's book collection.
    """
    collection = []

    def add(self):
        "Prompts the user for number and title."
        import classbook
        b = classbook.Book()
        b.create()
        self.collection.append(b)
```

The class `Catalog` imports the class `Book`.

# The Class Catalog

the file classcatalog.py

The collection of books is represented as a list,  
as the data attribute of the class `Catalog`.

```
class Catalog:
    """
    The class Catalog imports the class Book.
    It represents the library's book collection.
    """
    collection = []

    def add(self):
        "Prompts the user for number and title."
        import classbook
        b = classbook.Book()
        b.create()
        self.collection.append(b)
```

The class `Catalog` imports the class `Book`.

# The Class Catalog Continued

## 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

```
def show(self):
    "Shows the catalog."

def search_on_key(self, key):
    """
    Returns the book with the key if it is in
    the collection, else -1 is returned.
    """

def checkout(self):
    "Checks out the book with key."

def checkin(self):
    "Checks in the book with key."

def delete(self):
    "Deletes the book with key."
```

# The Class People

the file classpeople.py

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

```
class People:
    """
    The class People collects information of
    all librarians and patrons of the library.
    """
    def __init__(self):
        "Creates a root user."
        import classperson
        root = classperson.Person()
        root.create(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

the file classpeople.py

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

```
class People:
    """
    The class People collects information of
    all librarians and patrons of the library.
    """
    def __init__(self):
        "Creates a root user."
        import classperson
        root = classperson.Person()
        root.create(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

the file classpeople.py

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

```
class People:
    """
    The class People collects information of
    all librarians and patrons of the library.
    """
    def __init__(self):
        "Creates a root user."
        import classperson
        root = classperson.Person()
        root.create(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

## 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

```
def search(self,name):
    """
    Returns -1 if name not in self.whoswho,
    else the person object is returned.
    """
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.
    """
def who(self):
    "Shows who is currently logged in."
def logoff(self):
    "The current user is logged off."
def add(self):
    "Adds a new person to the collection."
def delete(self):
    "Prompts for a name and then deletes."
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *

def show_menu(p,c,w):
    "Shows the menu to the user."

def act(p,c,a,w):
    "Performs the requested action."

def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)

main()
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *
def show_menu(p,c,w):
    "Shows the menu to the user."
def act(p,c,a,w):
    "Performs the requested action."
def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)

main()
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *
def show_menu(p,c,w):
    "Shows the menu to the user."
def act(p,c,a,w):
    "Performs the requested action."
def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)
```

```
main()
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *
def show_menu(p,c,w):
    "Shows the menu to the user."
def act(p,c,a,w):
    "Performs the requested action."
def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)
```

```
main()
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *
def show_menu(p,c,w):
    "Shows the menu to the user."
def act(p,c,a,w):
    "Performs the requested action."
def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)
```

```
main()
```

# Main Program: libclassman.py

## 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

```
from classcatalog import *
from classpeople import *
def show_menu(p,c,w):
    "Shows the menu to the user."
def act(p,c,a,w):
    "Performs the requested action."
def main():
    "Main library management program."
    c = Catalog()
    p = People()
    w = -1
    print 'Welcome to our library!'
    while True:
        a = show_menu(p,c,w)
        if a == 9: break
        w = act(p,c,a,w)
```

```
main()
```

# Function show\_menu

## 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

```
def show_menu(p,c,w):  
    "Shows the menu to the user."  
    if w == -1: # no one logged on  
        print 'Please log on'  
        return 0  
    else: # w == 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 w == +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'  
        a = input('Make your choice : ')  
        return a
```

# Function show\_menu

## 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

```
def show_menu(p,c,w):
    "Shows the menu to the user."
    if w == -1: # no one logged on
        print 'Please log on'
        return 0
    else: # w == 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 w == +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'
        a = input('Make your choice : ')
    return a
```

# Function show\_menu

## 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

```
def show_menu(p,c,w):  
    "Shows the menu to the user."  
    if w == -1: # no one logged on  
        print 'Please log on'  
        return 0  
    else: # w == 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 w == +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'  
        a = input('Make your choice : ')  
        return a
```

# Function act

in file libclassman.py

```
def act(p,c,a,w):
    "Performs the requested action."
    r = w
    if a == 0:
        r = p.logon()
    elif a == 1:
        p.logoff()
        r = -1
    elif a == 2: c.show()
    elif a == 3: c.checkout()
    elif a == 4: c.checkin()
    elif a == 5: c.add()
    elif a == 6: c.delete()
    elif a == 7: p.add()
    elif a == 8: p.delete()
    return r
```

# Function act

in file libclassman.py

```
def act(p,c,a,w):
    "Performs the requested action."
    r = w
    if a == 0:
        r = p.logon()
    elif a == 1:
        p.logoff()
        r = -1
    elif a == 2: c.show()
    elif a == 3: c.checkout()
    elif a == 4: c.checkin()
    elif a == 5: c.add()
    elif a == 6: c.delete()
    elif a == 7: p.add()
    elif a == 8: p.delete()
    return r
```

## 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 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.
- 2 Write Python code for the class `Rational`.
- 3 Provide an `__init__` constructor method for the class `Catalog`, initializing the library with some books.
- 4 Modify the `create` in the class `Book` so that `title` and `key` are keyword variable arguments.
- 5 Describe how the design of our library manager would change if files would be used for the catalog and people. Which functions would change?