Lecture 21

Topics: *Chapter 10. Defining Classes*
10.4 Data processing with class
10.5 Objects and encapsulation
10.6 Widgets

10.4 Data processing with class

Example with dice from the previous meeting shows how useful a ***class*** can be for modeling a real world object with some behavior.

10.4 Data processing with class

Example with dice from the previous meeting shows how useful a ***class*** can be for modeling a real world object with some behavior.

Let's consider college students and their *grade point averages* (GPA).

In a typical college/university, courses are measured in terms of credit hours, and GPAs are calculated on a 4-point scale, with "A+" and "A" being 4 points, "A-" being 3.7 points, "B+" being 3.3 points, "B" being 3 points, etc.

10.4 Data processing with class

Example with dice from the previous meeting shows how useful a ***class*** can be for modeling a real world object with some behavior.

Let's consider college students and their *grade point averages* (GPA).

In a typical college/university, courses are measured in terms of credit hours, and GPAs are calculated on a 4-point scale, with "A+" and "A" being 4 points, "A-" being 3.7 points, "B+" being 3.3 points, "B" being 3 points, etc.

If a class is worth *3 credit hours* and the students gets an *"A,"* then he/she earns 3*4 = 12 *quality points*

GPA = total quality points / number of credit hours

10.4 Data processing with class

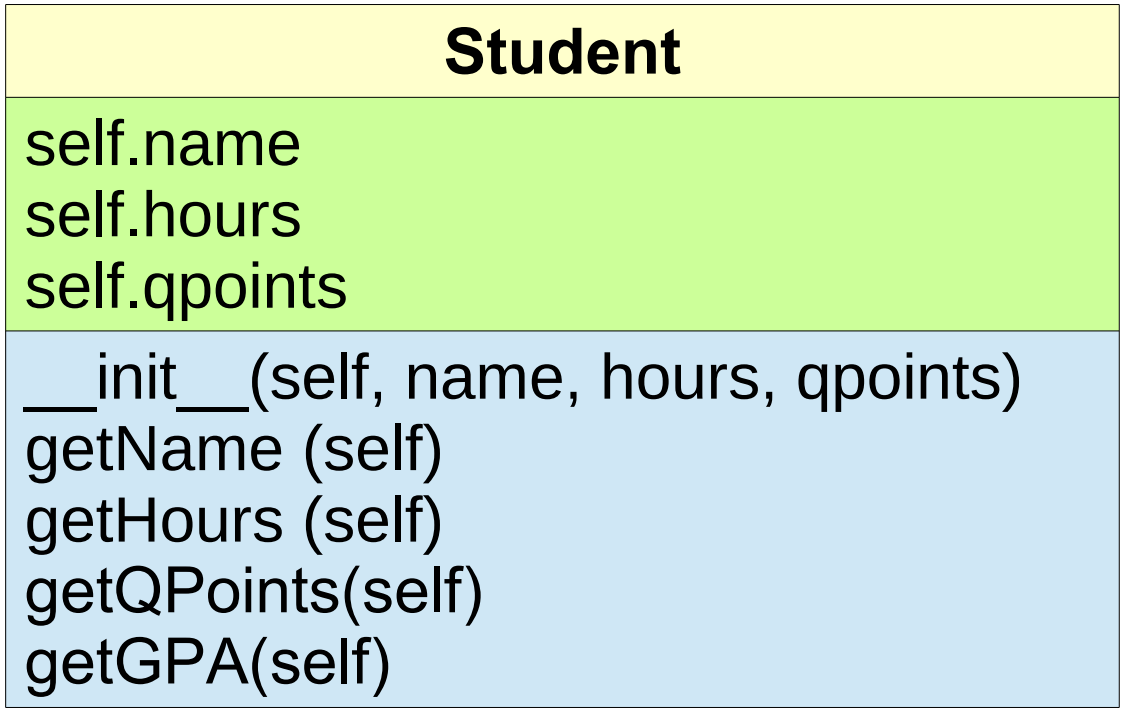
The data about students can be recorded into a file:

Adams, Samantha 56 222.32
Cole, Amanda 100 390
Jack, Adam 140 490
Katz, Mery 28 86.8
Zenith, Kevin 135 459

We will write a program that reads through this file to find the student with best GPA and print out his/her name, credit hours, and GPA.

10.4 Data processing with class

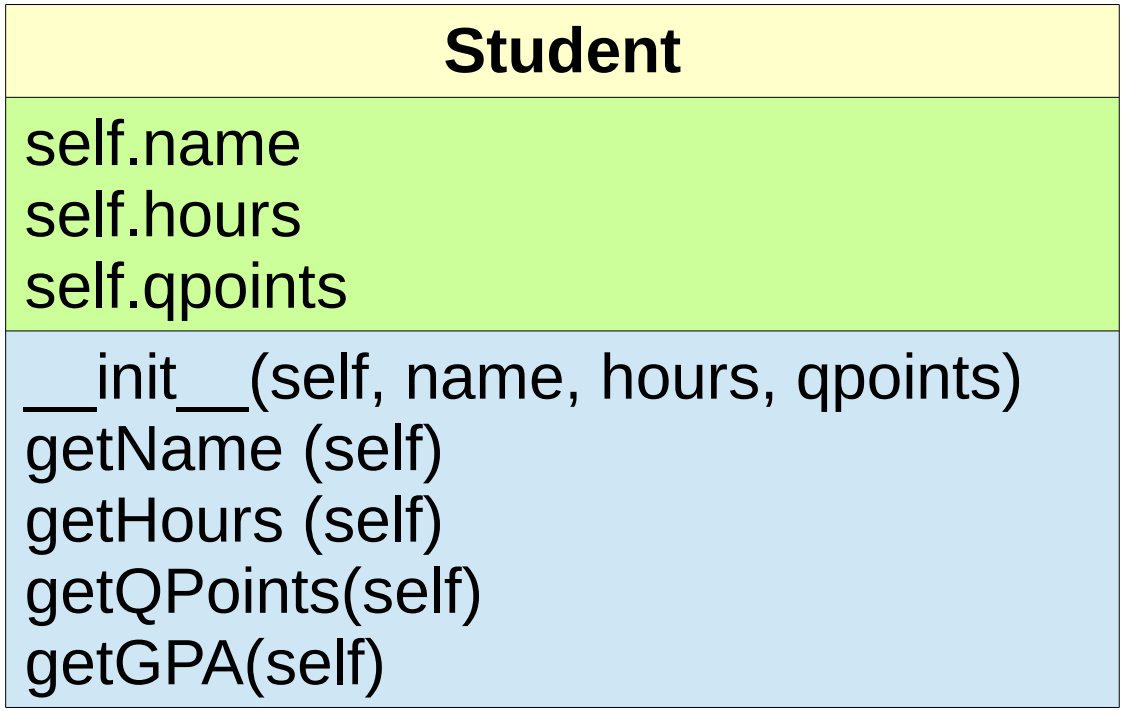
Student class:



class diagram

10.4 Data processing with class

Student class:



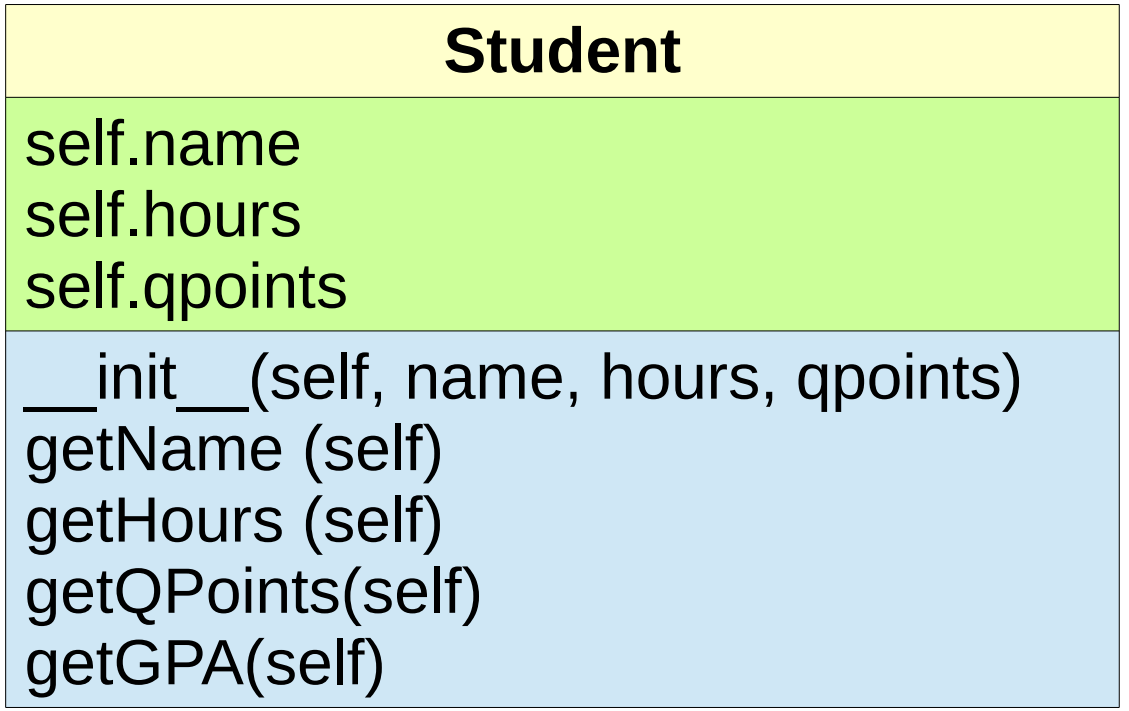
class diagram

Example of a Student instance:

```
personA = Student("Stone, Amelia",123,489.54)
```

10.4 Data processing with class

Student class:



class diagram

Example of a **`Student`** instance:

```
personA = Student("Stone, Amelia",123,489.54)
```


see program studentsGPA.py

10.5 Objects and encapsulation

Strategy:

identify useful objects → push the implementation details into a suitable class definition

then we can write an algorithm using these objects.

10.5 Objects and encapsulation

Strategy:

identify useful objects → push the implementation details into a suitable class definition

then we can write an algorithm using these objects.

This strategy gives us a separation of concerns (recall top-down design): we do not worry about object's implementation details, all we need to know is *what objects can do* (not how can they do it)

This separation of concerns is called *encapsulation*.

The implementation details of an object are *encapsulated* in the class definition.

10.5 Objects and encapsulation

Strategy:

identify useful objects → push the implementation details into a suitable class definition

then we can write an algorithm using these objects.

This strategy gives us a separation of concerns (recall top-down design): we do not worry about object's implementation details, all we need to know is *what objects can do* (not how can they do it)

This separation of concerns is called *encapsulation*.

The implementation details of an object are *encapsulated* in the class definition.

It is not enforced in Python, it is only a convention.

10.5 Objects and encapsulation

Putting classes in modules

A well-defined class can be used in many different programs.

Hence it is good to put it into a separate file and add some documentation that describes how the class can be used !

10.5 Objects and encapsulation

Putting classes in modules

A well-defined class can be used in many different programs.

Hence it is good to put it into a separate file and add some documentation that describes how the class can be used !

Type the following in Python interpreter:

```
>>> import random
>>> print(random.random.__doc__)
random() -> x in the interval [0, 1).
```

and this:

```
>>> help(random.random)
Help on built-in function random:

random(...) method of random.Random instance
    random() -> x in the interval [0, 1).
```

10.5 Objects and encapsulation

```
def makeStudent(line):
    """ creates an instance of class Student;
    line is a line from the file formatted as
    LastName, FirstName  hours  qpoints """

    name, hours, qpoints = line.split("  ")
```

docstring

Docsrings are used to get help (brief description) about class/method/module.

10.5 Objects and encapsulation

Recall our `MSDie` class. Let's play more with it!

- put the definition of the `MSDie` class into a separate file (module) called die.py.

- the main function will be in the file program.py

- the die.py module will be imported in the program.py by `from die import *`

- let's define a class for buttons: `Button` and save it in module button.py

- let's define a class for display the die: `dieView` and save it in module dieView.py

10.5 Objects and encapsulation

MSDie	
self.sides	self.value
__init__(self, sides)	roll (self)
getValue (self)	
setValue(self, value)	

dieView	
self.win	self.sides
self.display	
__init__(self, win, center)	
show (self)	hide (self)
setValue(self, value)	

Button		
self.win	self.br	self.bt
self.x_min	self.x_max	self.y_min
self.y_max	self.active	
__init__(self, win,center,width=70,height=40,label="EXIT")		
activate (self)	deactivate (self)	
clicked(self, p)	getLabel(self)	

were added later on

10.6 Widgets

GUI stands for Graphical User Interface

GUI is usually composed of *visual interface objects*, called *widgets*.

`Entry` object from graphics library is a *widget*.

`DieView` we defined, is a *widget*.

`Button` we defined, is also a *widget*.