CSC148 L5102

Introduction to Computer Science
Joe Lim
joe.lim@utoronto.ca

Outline

- Special methods, managed attributes
- Types within types …
- Generalized classes with inheritance

Attributes and Properties

- Ability or characteristics which something or someone has
- Binding attributes to objects – general concept in Python
- Method differs from function in 2 aspects:
  - *Belongs to a class and it is defined within a class*
  - *First parameter has to be a reference to the instance which called the method. – "self"*

Encapsulation

- Data Encapsulation – bundling o data with the methods that operate on the data
  - *Achieve by getter and setter methods*
- Information hiding – some internal information or data is "hidden", so that it can't be accidentally changed
- Data Abstraction = Data Encapsulation + Data Hiding

Encapsulation

- _name → Protected → Not used outside of class definition
- __name → Private → Inaccessible & Invisible, can't write or read to these attributes unless inside the class
- Attributes of a class are made private to hide and protect them from other code

Getter/Setter Methods

- Getter → Retrieve data
- Setter → Modify data

Rational Fractions

Although python as a built-in type for floating-point numbers, there is no built-in type for representing rational numbers:

Rational numbers are ratios of two integers p/q. where p is called the numerator and q is called the denominator. The denominator q is non-zero. Operations on rational fractions include addition, multiplication, and comparisons:

>, <, >+, <=, =

… so we'll have to create our own Rational class.

Building class Rational

Define a class API:
1. Choose a class name and write a brief description in the class docstring.
2. Write some examples of client code that uses your class
3. Decide what services your class should provide as public methods, for each method declare an API[1] (examples, header, type contract, description)
4. Decide which attributes your class should provide without calling a method, list them in the class docstring

[1]Use the CSC108 Function Design Recipe

…class Rational

Implement the class:
1. Body of special methods __init__, __eq__, and __str__

2. Body of other methods

Special, aka magic, methods

- Python recognizes the names of special methods such as __init__, __eq__, __add__, and __mul__ and has short-cuts (aliases) for them. This syntactic sugar doesn't change the semantics (meaning) of these methods, but may allow more manageable code.

- For example, suppose you create a list of Rational, and then want to sort it, or check to see whether an equivalent element is in it …

Managing Attributes num and denom

Suppose that client code written by billions of developers uses Rational, but some of them complain that the class doesn't protect them from silly mistakes like supplying non-integers for the numerator or denominator, or even zero for the denominator …

After you have already shipped class Rational, you can write methods **get_num**, **set_num**, **get_denom**, and **set_denom**, and then use **property** to have python use these functions whenever it sees **num** or **denom**

Inheritance

- Class can inherit attribute & behaviour methods from another class, called the superclass.
- For example:
  - *A class called Person has 2 attributes, firstname and lastname*
  - *It has only one method, the Name method*
  - *Employees are Persons in Companies*
  - *If we create a class called Employee without inheriting from class Person, we have to define all attributes and methods in the Employee class*

Clock, Calendar and ClockCalendar

- More Examples of inheritance
- Create a class Clock which has the ability to tell time and will increment the time for each tick
- Create a class Calendar which has the ability to give you the date, be able to let you know leap years and can be advanced to the next day manually.
- Create a class ClockCalendar which inherits from both Clock and Calendar and has the ability to give you and time and date.

Shapes with extras

Devise the following class…

Squares have four vertices (corners) have a perimeter, an area, can move themselves by adding an offset point to each corner, and can draw themselves.

Use composition

Squares need drawing capabilities, so make sure each Square has a Turtle. Furthermore, the vertices of Squares are Points, and if we include those we'll get the ability to add an offset point and calculate distance… All without writing code to duplicate the capabilities of Turtle or Point.

More Square-like classes

What if we decided to devise a RightAngleTriangle class with similar characteristics to Square?

There is an implementation of RightAngleTriangle, but it has a problem:

There's a lot of duplicate code. What do you suggest?

We could try …

- Cut-paste-modify Square → RightAngleTriangle?
- Include a Square in the new class to get at its attributes and services?

We really need a general Shape with the features that are common to both Square and RightAngleTriangle, and perhaps other shapes that come along

Abstract class Shape

- Most of the features of Square are identical to RightAngleTriangle.
- The differences are the class names (Square, RightAngleTriangle) and the code to calculate the area.
- Put the common features into Shape, with unimplemented _set_area as a place holder …
- Declare Square and RightAngleTriangle as subclasses of Shape, inheriting the identical features and over-riding _set_area