CSC 148 Winter 2017

Week 2

Special methods, property, composition, inheritance

Bogdan Simion

bogdan@cs.toronto.edu

http://www.cs.toronto.edu/~bogdan

University of Toronto, Department of Computer Science

Announcements

- Computing help
  - email computinghelp@teach.cs.toronto.edu

    or post on the computing help discourse board

    https://bb.teach.cs.toronto.edu/c/computing-help
  - Hours: M1-3 (MAC help as well), M5-6, T3-5, W1-3 (MAC help as well), W4-6, Th12-2
- Slides – posted weekly
- Recognized Study Groups (see course webpage)

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”

Special methods

- Rational
  - special methods

Interesting aspects of Python

- What if I try these?

```
print(Point.x)
Point.y = 17
```

- Class namespaces vs object namespaces
- Methods can be invoked in two equivalent ways:

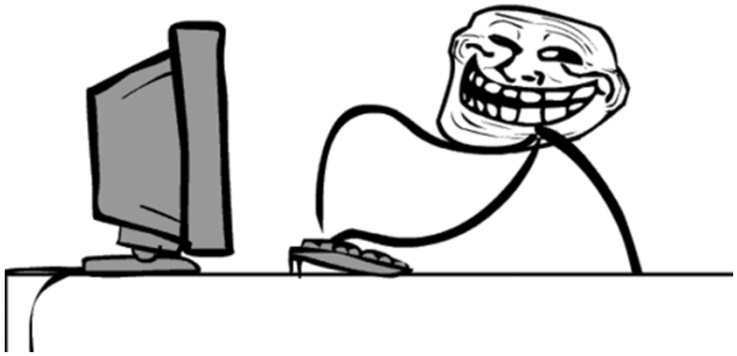
```
p = Point(3, 4)
p.distance_to_origin()
5.0
Point.distance_to_origin(p)
```

  - *In both, the first parameter (self) refers to the instance named p*

- Don't be afraid to explore (yes, even seemingly weird code) and try to analyze and explain the result!

Protecting against mistakes

- Bad inputs can cause programs to crash



- What if num and denom are not integers?
- What if denom is 0?

Announcements

- Anonymous feedback – really helpful!
- Note-takers – volunteers needed

UNIVERSITY OF TORONTO

ACCESSIBILITY SERVICES

student life.

Accessibility Services Note-Taking Program

Volunteer Note-Takers Needed!

Why volunteer as a note-taker?

- Help your fellow students living with a disability to achieve academic success
- Many volunteers report that while helping others their own note-taking skills improve
- You will receive a Certificate of Appreciation upon the completion of the semester
- You will receive CCR (Co-Curricular Record) recognition

Does it take up a lot of my time?

Volunteering as a note-taker is easy and involve almost no extra work:

- Continue to attend classes regularly and take lecture notes
- Upload your notes to our secure website consistently
- Inform us if you drop any classes that you volunteered

How to volunteer as a note-taker?

Step 1: Register as a volunteer note-taker online at:

https://sites.studentlife.utoronto.ca/accessibility/vollogin.aspx

Step 2: Select your course and click register

Step 3: Upload your notes after every class

- Typed notes can be submitted online
- Legible hand-written notes can be scanned and uploaded at our office (or you can do it at home if you have a scanner)

Thank you for your generous help!

Email us at: as.notetaking@utoronto.ca or call 416-978-6186 if you have questions or require any assistance.

Our Address: Accessibility Services

455 Spadina Avenue, 4th Floor, Room 400

Office Hours: 9:30 AM - 4:30 PM (Monday to Friday)

Daily Office closure 12:30 PM - 1:30 PM

Next up ...

- Managed attributes, properties
- Types within types ... composition!
- Generalize classes with inheritance

Data encapsulation

- Data encapsulation (aka Data hiding) == implementation details of a class are kept hidden from the user

- The user should only perform a restricted set of operations on the "hidden" members of the class, through special methods
  - This is where getter and setter methods come in (will see these in a bit)

Encapsulation - Conventions

- Python - conventions:
  - _name => Should not be used outside of class definition
    - Convention, not enforced by Python
  - __name => Inaccessible & Invisible (in theory anyway, there is actually a way to access them in Python)
    - Can't read or write to these attributes unless inside the class

- Attributes of a class are made private to hide implementation details or protect them from unauthorized use
  - Basically, only allow reading and writing to them via special methods that the class designer can control

Getters, setters, and properties

- A way to turn public attributes into private ones gracefully
- Basic idea: make accesses (read, write) to public attributes go through special getter and setter methods
- Example: ...

OOP features - overview

- Composition and inheritance
  - A square has some vertices (points), so does triangle, etc.
  - A square is a shape, so is a triangle, etc.
- Relationship between has_a and is_a
- Abstraction
  - A shape has a perimeter (triangle, square, etc.)
    - A square or triangle can inherit the perimeter from a shape
  - A shape has an area (triangle, square, etc.)
    - Can area be abstracted at the shape level?

Composition

- Use existing types inside new user-defined types

Example

- Say we want to implement class Square:

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

Example

- Say we want to implement class Square:

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

Composition

- We need:
  - Ability to draw a Square => each Square needs a Turtle
  - Vertices, aka Points => need Point to represent corners
    - We also get the Point's "abilities": to move by an offset, to calculate a distance, etc.
  - Composition allows us to avoid writing code to duplicate the abilities of Turtle and Point

- Implementation ...

More Square-like classes

- Say we want to implement class RightAngleTriangle:

Right-angled triangles have three vertices (corners), have a perimeter, an area, can move themselves by adding an offset to each corner, and can draw themselves.

More Square-like classes

- Say we want to implement class RightAngleTriangle:

Right-angled triangles have three vertices (corners), have a perimeter, an area, can move themselves by adding an offset to each corner, and can draw themselves.

- Sounds very similar, right?
- Implementation ... Options?

We could try ...

1. Copy-paste-modify Square => RightAngleTriangle

... that's a lot of duplicate code though!

2. Include a Square in the new class to get at its attributes and services?

We really need a general Shape with common features to both Square and RightAngleTriangle (and possibly others)

Abstract class Shape

- Most features of Square are identical to RightAngleTriangle
  - Corners/points, perimeter, area, move, draw
- Differences: class name, code to calculate the area

- Key idea: Place common features into class Shape, with unimplemented _set_area as a place-holder...

- Declare Square and RightAngleTriangle as *subclasses* of Shape, *inheriting* the identical features by declaring

```
Class Square(Shape): ...
```

Developing Shape, Square ...

Inherit, override, or extend?

- Subclasses use three approaches to recycling the code from their superclass, using the same name
  - 1. Methods and attributes that are used as-is from the superclass are inherited – example?
  - 2. Methods and attributes that replace what's in the superclass are overriden – example?
  - 3. Methods and attributes that add to what is in the superclass are extended – example?

Write general code

- Client code written to use Shape will now work with subclasses of Shape – even those written in the future.

- The client code can rely on these subclasses having methods such as move_by and draw

- Here is some client code that takes a list of objects from subclasses of Shape, moves each object around, and then draws it.