CSC148 L5102

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

Introduction

- Joe Lim
- Email: joe.lim@utoronto.ca - Prefix CSC148 on the subject line
- Office Hours: Wednesdays, 5-6pm PB150

You?

- Major or Specialist?
- Graduate School?
- Completed 108?
- Knowledge of Python?
- Knowledge of OOP?

Course Info

- Course Website:
  - http://www.cdf.toronto.edu/~csc148h/winter/
  - *All course materials will be provide*
  - *No textbook!*
- Submission - Markus
- Discussion Board – Piazza
- Read and understand Course Info Sheet

Tutorials (Labs)

- If you are registered in Friday 7-9 or Thursday 7-9 tutorial, please consider moving to Thursday 9, 11, 1 or 3 tutorials. More spaces are added to these.

Academic Integrity

- All submissions must present original, independent work.
- We take academic offenses very seriously.
- Your goal is to learn. No one learns by cheating!
- Please read
  - *"Guideline for avoiding plagiarism"*
    http://www.cs.toronto.edu/~fpitt/documents/plagiarism.html
  - *"Advice about academic offenses"*
    http://www.cs.toronto.edu/~clarke/acoffences/

Course Work

Work	Due	Weight
9 Labs	Every Week Except Wk1 and Wk 12	9%
2 Assignments	A1 - February 25th, 10pm A2, March 24th, 10pm A2, Demo/Interview during lab March 31st	21%
SLOG	Wk3 – Wk12	6%
2 Term Tests	T1 – February 10th during lecture time T2 – March 16th during lecture time	26%
Final Exam	TBA	38%

Marking Scheme

- Assignment
  - *14% - best work, 7% - worst work*
- Term Test
  - *16% - best effort, 8% - worst effort*
  - *1% per test for completing a test review exercise*
- Final Exam
  - *Need to get 40% to pass the course*

CSC148 Ramp-up Session

- No Python experience?
- Want to brush up your Python skill
- Send email to csc148w16rampuo@cs.toronto.edu
- There is ONLY space for 240 students
- Session will be WB116 (Wallberg Bldg), Saturday, January 16th, 10am – 4pm

So, what is CSC148 about?

- Understanding and writing a solution for a real-world problem
- Abstract Data Types (ADTs) to represent and manipulate information
- Recursion: Calling yourself
- Exceptions: - Dealing with unexpected situations
- Design: - Program structure
- Efficiency: Resource (time/space) used by program

Objects and Classes

- What are objects?
  - *Unique instance of a data structure modelling a real world concept.*
- What are classes?
  - *User-defined prototype for an object*
- Objects are working instances of a class

Objects in Python

- Everything in Python is an object
  - *str, int, list, dict, etc*

Point

Somewhere in the real world there is a description of points in two-dimensional space:

In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Find the most important noun (good candidate for a class. . .), its most important attributes, and operations that sort of noun should support.

Building class Point

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 Point

Implement the class:

1. Body of special methods __init__, __eq__, and __str__
2. Body of other methods
3. Testing (more on this later)

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.

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**

Point for thoughts...

- What happens if, after declaring Point, you try
  print(Point.x)
  OR
  Point.y = 17
- Methods can be invoked in two equivalent ways:
  p = Point(3, 4)
  p.distance_to_origin()
  5.0
  Point.distance_to_origin(p)

  In each case the first parameter, conventionally, self, refers to the I nstance named p