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 provided
- 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”
  - “Advice about academic offenses”
    http://www.cs.toronto.edu/~clarke/academicoffenses/
Course Work

<table>
<thead>
<tr>
<th>Work</th>
<th>Due</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 Labs</td>
<td>Every Week Except Wk1 and Wk 12</td>
<td>9%</td>
</tr>
<tr>
<td>2 Assignments</td>
<td>A1 - February 25th, 10pm A2 - March 24th, 10pm A3 - March 31st</td>
<td>21%</td>
</tr>
<tr>
<td>SLOG</td>
<td>Wk3 - Wk12</td>
<td>6%</td>
</tr>
<tr>
<td>2 Term Tests</td>
<td>T1 - February 10th during lecture time T2 - March 16th during lecture time</td>
<td>26%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>TBA</td>
<td>38%</td>
</tr>
</tbody>
</table>

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
### 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 (examples, header, type contract, description).
4. Decide which attributes your class should provide without calling a method, list them in the class docstring.

---

### 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:

\[
>, <, \geq, \leq, =
\]

...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-integer 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.x = 17
- Methods can be invoked in two equivalent ways:
  \[
p = \text{Point}(3, 4)
  \]
  \[
p = \text{get_distance_to_origin}(p)
  \]
  In each case the first parameter, conventionally set, refers to the instance named p