CSC148 winter 2017

Introduction to computer science
week 1

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~csc148h/winter/
416-978-5899

reading:
http://www.cdf.toronto.edu/~csc148h/winter/148Notes.pdf

January 11, 2017

Outline

What's CSC148 **not** about?

- well first, CSC108 was about `if` statements, loops, function definitions and calls, lists, dictionaries, searching, sorting, classes, documentation style. So you've got all that down...

  The sessions will be in BA1180 (BA room 1180) Saturday and Sunday January 7th and 8th. There is space for about 120 per day, and you need to register

But what's CSC148 about?

- how to understand and write a solution for a real-world problem
- abstract data types (ADTs) to represent and manipulate information
- recursion: clever functions that call themselves
- exceptions: how to deal with unexpected situations
- design: how to structure a program
- efficiency: how much resource (time/space) does a program use?

How's this course run?

All answers in course information sheet. Spoiler alert: meaning of life is 42...

review function design recipe

Dream up a function. Now use the function design recipe to build it, step-by-step... Now with PyCharm

python infested by objects

Here are some built-in objects to fool around with:

```
>>> w1 = "words"
>>> w2 = "swords"[1:]
>>> w1 is w2
>>> w1 == w2
>>> w1 * w2
>>> import turtle
>>> t = turtle.Turtle()
>>> t.pos()
(0.00,0.00)
>>> t.forward(100)
```

vandalizing existing classes

this is deeply wrong, except for teaching purposes...

```
>>> from turtle import Turtle
>>> t1 = Turtle()
>>> t1.pos()
(0.00,0.00)
>>> t1.forward(100)
>>> t1.pos()
(100.00,0.00)
>>> t1.neck
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Turtle' object has no attribute 'neck'
>>> Turtle.neck = "very reptilian"
>>> t1.neck
'very reptilian'
```

Design a new class

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.

build class Point...

in that deeply wrong, but informative, way

```
>>> from math import sqrt
>>> class Point:
...    pass
...
>>> def initialize(point, x, y):
...     point.x = x
...     point.y = y
...
>>> def distance(point):
...     return (point.x**2 + point.y**2) ** (1 / 2)
...
>>> Point.__init__ = initialize
>>> Point.distance = distance
>>> p2 = Point(12, 5)
>>> p2.distance()
13.0
>>>
```

build class Point. . . properly!

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 you class should provide without calling a method, list them in the class docstring

[1] use the CSC108 function design recipe

continue building class Point. . . properly!

Implement the class:

1. body of special methods `__init__`, `__eq__`, and `__str__`
2. body of other methods
3. testing (more on this later)

weird things

- 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 instance named **p**