CSC148 winter 2014
inheritance, Exceptions, special methods
week 3

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

January 24, 2014
Outline

specialize software

raising exceptions
specialize flexibly

If we decided to extend the features of Stack, what’s wrong with:

- modifying the existing Stack?
- cut-paste-modify Stack → MyStack?
- include Stack attribute in new classes
we subclass (extend) a superclass (base class) by:

- declaring that we’re extending it...
  
  class NewClass(OldClass):
    ...

- add methods and attributes to specialize

- other methods and attributes are searched for in superclass
override versus extend

you may replace or modify old code

- subclass method with the same name replace superclass method

- access superclass method with
  `OldClass.method(self,...)`

- `__init__` is a special case — careful
return types are not appropriate in all cases

- what’s wrong with IntStack returning a “special” integer for pop-on-empty?

- push usually has return type None, but what if stuff happens?

- what if the calling code doesn’t know what to do?
cause existing Exceptions:

- `int("seven")`
- `a = 1/0`
- `[1, 2][2]`
raise existing Exceptions:

- raise ValueError or...

- raise ValueError("you can’t do that!")
roll your own Exceptions:

- class ExtremeException(Exception):
  pass

- raise ExtremeException

- raise ExtremeException('I really take exception to that!')
what makes two stack equivalent?

Tell Python with \_\_eq\_\_

Your \_\_eq\_\_ should really be equivalent: symmetrical, reflexive, transitive
represent in a reproducible way

Tell Python how to represent your object with \texttt{\_\_repr\_\_}

Ideally, you should be able to cut-and-paste this representation to create an equivalent object