CSC148 winter 2015

inheritance, Exceptions week 3

Danny Heap heap@cs.toronto.edu BA4270 (behind elevators)

http://www.cdf.toronto.edu/~csc148h/winter/ 416-978-5899

January 19, 2015





Outline

specialize software

raising exceptions

idiomatic python

recursion





specialize flexibly

If we decided to extend the features of Shape, what's wrong with:

modifying the existing Shape?

ightharpoonup cut-paste-modify Shape \longrightarrow MyShape?

▶ include Shape attribute in new classes



class declaration

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



how python finds methods

For each name (method or attribute) Python looks first in the class of this instance, then in the superclass, then further.

Try tracing this

abstract superclass

Suppose I want a collection of classes to guarantee the same behaviour. If the behaviour is declared in their common superclass, then:

```
tur = turtle.Turtle()
t = Triangle(0, 0, 20)
import square
s = square.Square(0, 0, 20)
shapes = [t, s]
for i in shapes:
    for j in range(5):
        i.position = (j * 10, j * 10)
        i.draw(tur)
```



richer communication

return types are not appropriate in all cases

what's wrong with Stack 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")

= 1/0

▶ [1, 2][2]

raise existing Exceptions:

▶ raise ValueError or...

raise ValueError("you can't do that!")

roll your own Exceptions:

Lass ExtremeException(Exception):
pass

raise ExtremeException

raise ExtremeException('I really take exception
to that!')



going with the (pep) tide

Python is more flexible than the community you are coding in. Try to figure out what the python way is

- don't re-invent the wheel (except for academic exercises), e.g. sum, set
- ▶ use comprehensions when you mean to produce a new list (tuple, dictionary, set, ...)
- ▶ use ternary if when you want an expression that evalutes in different ways, depending on a condition



example: add (squares of) first 10 natural numbers

➤ You'll be generating a new list from range(1, 11), so use a comprehension

➤ You want to add all the numbers in the resulting list, so use sum

list differences, lists without duplicates

▶ python lists allow duplicates, python sets don't

python sets have a set-difference operator

python built-in functions list() and set() convert types





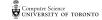
re-use and recursion — take one!

▶ a function sum_list that adds all the numbers in a nested list shouldn't ignore built-in sum

...except sum wouldn't work properly on the nested lists, so make a list-comprehension of their sum_lists

but wait, some of the list elements are numbers, not lists!

write a definition of sum_list — don't look at next slide yet!





hey! don't peek!

```
def sum_list(L):
    """ (list) -> float
    Return sum of the numbers in possibly nested list L
    >>> sum_list([1, 2, 3])
    6
    >>> sum_list([1, [2, 3, [4]], 5])
    15
    11 11 11
    return sum( # sum the elements of list...
               # if x is a sublist, sum_list(x)
                [sum_list(x) if isinstance(x, list)
                             else x # if not list, then number
               for x in Ll)
```

tracing recursion

To understand recursion, trace from simple to complex:

- ▶ trace sum_list([1, 2, 3]). Remember how the built-in sum works.
- ▶ trace sum_list([1, [2, 3], 4, [5, 6]]). Immediately replace calls you've already traced (or traced something equivalent) by their value
- ▶ trace sum_list([1, [2, [3,4], 5], 6 [7, 8]]). Immediately replace calls you've already traced by their value.



sample solutions

▶ trace sum_list([1, 2, 3]). Remember how the built-in sum works.

Solution: sum([1, 2, 3]) = 6

- ▶ trace sum_list([1, [2, 3], 4, [5, 6]]). Immediately replace calls you've already traced (or traced something equivalent) by their value Solution: sum([1, 5, 4, 11]) = 21. We already knew what sum_list does with a flat list like [2,3] or [5, 6]
- ▶ trace sum_list([1, [2, [3, 4], 5], 6 [7, 8]]). Immediately replace calls you've already traced by their value.

Solution: sum([1, 14, 6, 15]) = 36. We already know what sum_list does with nested lists like [2, [3, 4], 5]

