

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?
- ▶ 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



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)
```



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!')`



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`



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
```

```
"""
```

```
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 L])
```



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]`