CSC148 fall 2013
names, tracing, abstraction, recursion
week 12

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

November 28, 2013
Outline

memory model

tracing... or not

consequences of recursion
how much detail for developers?

Enough detail to predict results and efficiency of our code — more detail than end users, less than compiler/interpreter designers. In Python:

- **Every name contains a value**

  What if a name is used 2 or more times?

- **Every value is a reference to the address of an object**
Searching for names

Python looks, in order:

- Innermost scope (function body, usually) local
- Enclosing scopes nonlocal
- Global (current module or __main__)
- Built-in names
- See scopes and namespaces
intense example

Try running `python docs namespace example` to check your comfort level
The first parameter, conventionally called `self`, is a reference to the instance:

```python
>>> class Foo:
...     def f(self):
...         return "Hi world!"
...     
>>> f1 = Foo()

Now `Foo.f(f1)` means `f1.f()`
hunting for a method...

Start in the nearest subclass and work upwards, for example visualize method
def rec_max(L):
    """
    Return the maximum number in possibly nested list of numbers.
    >>> rec_max([17, 21, 0])
    21
    >>> rec_max([17, [21, 24], 0])
    24
    >>> rec_max([17, [21, 24], [18, 37, 16], 0])
    37
    """
    return max([rec_max(x) if isinstance(x, list) else x for x in L])

Recommended:
- trace the simplest (non-recursive) case
- trace the next-most complex case, plug in known results
- same as previous step...
TMI tracing

In contrast to the step-by-step, plus abstraction (previous slide), you could trace this in the visualizer
TMI tracing

In contrast to the step-by-step, plus abstraction (previous slide), you could trace this in the visualizer.