We learn about program decoration.

1. Questions?

2. A decorator is a function that modifies the definition of another function or class.
   
   (a) We have seen a function decorator with @property. This decorator causes a method to appear to be an attribute.

   (b) We have seen a class decorator with @total_ordering. This decorator writes comparison methods (e.g. __gt__) that the user was too lazy to write for classes that can order any pair of objects. @total_ordering is a decorator is imported from the functools package.

3. We can specify our own decorators.
   
   (a) Typing

   ```python
   @D
def f():
   ...
   ```

   is exactly the same as writing

   ```python
def f():
   ...
   ```

   ```python
   f = D(f)
   ```

   (b) An important observation: when we define a function B inside another function, A, every time we call A we get a brand new definition of B. That definition is local to A and cannot be seen outside of A.

   (c) Remember that we can pass a reference to a function around by just using its name.

   (d) Now we can build a decorator that prints beep every time the decorated function is called:

   ```python
   def beep(f):
   def _f():
       print("beep")
       return f()
   return _f
   ```

   (e) When we apply this decorator to a function, the definition gets modified by beep:

   ```python
   >>> @beep
   ... def yo():
   ```
... print("Hello, world.")
...
>>> yo()
beep
Hello, world.

(f) We can write a decorator that counts function calls:

(g) We can write a memoizer: