We learn about program decoration.

1. Questions?

2. Recall: A decorator is a function that modifies the definition of another function or class.

3. Previously we saw the decorator, memoize:

```python
def memoize(f):
    """Wrap f with a cache of function return values.""
    def wrapper(*args):
        if args in wrapper.cache:
            wrapper.hits += 1
            return wrapper.cache[args]
        else:
            wrapper.misses += 1
            result = f(*args)
            wrapper.cache[args] = result
            return result
    wrapper.cache = dict()
    wrapper.hits = 0
    wrapper.misses = 0
    return wrapper
```

The function wrapper that was defined locally and on-the-fly is called a closure because it captures the value of f. We might use this in

```python
@memoize
def fibo(n):
    """Compute n-th fibonacci number""
    if n <= 1:
        return 1
    else:
        return fibo(n-2)+fibo(n-1)
```

The dictionary or cache that memoize uses to avoid duplicate computation causes fibo to be computed using dynamic programming.

4. Exercise: Suppose postage stamps cost 49 cents for a letter and 34 cents for a postcard. What is the smallest number of penny/postcard/letter stamps that can be bought with n cents?
5. Some new “special” attributes for objects:

(a) The \_name\_ attribute is a string that represents the declared name of the object. For example, `math.sin._name_` is 'sin'.

(b) The \_module\_ attribute is a string that represents the module name of the declared object. `math.sin._module_` is 'math'.

(c) The \_doc\_ attribute is a string that represents the documentation comment associated with the object. `math.sin._module_` is 'Return the sine of x (measured in radians).'

6. The `@wraps` decorator factory.

(a) Notice what happens when we ask `fibo` for its name. Notice, also that the `doctest.testmod` system fails to run tests on `fibo`.

(b) We can fix this by writing `update_wrapper(wrapper, wrapped)`. This function assigns the wrapper function the name, module, and comment attributes of the wrapped function.

(c) We then build a `factory` for decorators. Each decorator changes the identity of the decorated function to be the identity of the function captured by the decorator.

(d) Both `update_wrapper` and `@wraps` are part of the `functools` package.

7. The `checkdoc` class decorator.

(a) When we're writing classes, it's helpful to check to make sure that each defined method of the class has a documentation string.

(b) The `dir(C)` function returns methods associated with the class. We simply walk through the resulting list and verify that the \_doc\_ attribute of each method is not `None`. Tricky point: the class does not yet have a name.

8. If an object, `o`, has a \_calls\_ method, that method is used when you want to “call” it: `o(x)` is really `o._calls_`(x).

*