Model-View-Controller and first class functions

Lists of functions

```python
class Shopping:
    def __init__(self):
        self.to_buy = []

    def remember_to_buy(self, item):
        self.to_buy.append(item)

    def go_shopping(self):
        for item in self.to_buy:
            item()

    def apples(self):
        print("Buy some apples")

    def milk(self):
        print("Buy a half gallon of milk")

groceries = Shopping()
groceries.remember_to_buy(apples)
groceries.remember_to_buy(milk)
groceries.go_shopping()
```

Why would you treat functions as objects?

One reason: To invert dependence

Object X needs to call a method in Y

I don’t want X to “know about” (depend on) Y

Example: I don’t want Sudoku board to “know about” how it is displayed on the screen

Is that a thing?

```python
def shout():
    print("Hey!")

def do_it(f):
    f()

do_it(shout)
```

Yes, it is.
Sudoku board display

When we find a duplicate square, I want to report it.
Maybe by printing a message.
Maybe by highlighting it in red.
Maybe something else ...
   I may want to add options later, without changing the Sudoku board logic.

Boggler code was tangled ...

```python
def mark_taken(row, col):
    """ Marks the tile at row,col as currently in use ... ""
    global in_use
    assert row >= 0 and row < len(content)
    assert col >= 0 and col < len(content[0])
    assert in_use[row][col] == False
    in_use[row][col] = True
    grid.fill_cell(row, col, grid.green)
    grid.label_cell(row, col, content[row][col])
```

Disentangling ...

I want to separate the logic ("model") from the display ("view")

But somehow we must notify the view when the model changes, as when a duplicate tile is found

Solution: Functions as data
   Call them without knowing their names
class Tile:

```python
def __init__(self, row, col, sym):
    ...
    self.listeners = []
    ...

def register(self, listener):
    self.listeners.append(listener)

def announce(self, event):
    for func in self.listeners:
        func(self, event)
```

```python
in sudoku.py:

def handle_events(tile, event):
    if event == "duplicate":
        grid.fill_cell(tile.row, tile.col, color=grid.red)
        grid.label_cell(tile.row, tile.col, tile.symbol, color=grid.white)

def display(board):
    grid.make(9,9,500,500)
    for row in range(9):
        for col in range(9):
            tile = board.tiles[row][col]
            tile.register(handle_events)
            grid.fill_cell(row, col, grid.white)
            grid.label_cell(row, col, tile.symbol)
```

```python
in sdkdisplay.py:

def display(board):
    grid.make(9,9,500,500)
    for row in range(9):
        for col in range(9):
            tile = board.tiles[row][col]
            tile.register(handle_events)
            grid.fill_cell(row, col, grid.white)
            grid.label_cell(row, col, tile.symbol)
```

Optional attachment of graphic display, depending on the command line
Using call-backs to break dependence

“Register” a function:
Store it in a list, in a module that doesn’t know what it does or where it lives

“Notify” the “listener”:
Call a stored function when an event occurs

Ex: “announce” method in Tile calls each registered listener with a “duplicate” event

Where else will you see this?

Hardware: device drivers use call-backs

• So your computer operating system doesn’t need to know the details of your disk drive, graphics card, etc

Graphical user interface toolkits

• Register call-backs for mouse motion, keyboard presses, window uncovered, ...
• Javascript: Handlers for events on web pages

Event-oriented parsers

• SAX model XML readers: Handlers for each node type

Architectural design patterns

We had an architectural problem:

A dependence that entangled program logic with display. It wasn’t a problem of functionality, but of understandability and maintainability.

It’s a standard problem ... maybe it has a standard solution.
It does. That’s called a pattern.

People build catalogs of design patterns, and give them standard names so we can talk about them. You’ll learn more of them in CIS 211.