Announcements

- Homework
  - HW8 is out! But due date postponed.
  - You should be able to do everything on it after today though regardless.
- We'll talk about the upcoming midterm here in a moment
- Friday is the last day to choose C/NC for any class
  - If you want and have questions/concerns, I am happy to chat about it with you
- Polling: `rembold-class.ddns.net`

Midterm Plans

- Given that people are now scattered across the nation (and globe), it doesn't make much sense to try to hold a synchronized test
- I still need feedback about what you have learned, so we are switching to a mini-project model
- Deliverables due on Friday night:
  - 1-2 python scripts of your own devising that do something concrete (ie. they have a specific objective that they deliver on)
  - A short write-up for each script.
- I'm pushing back HW8 to be due at the same time as HW9 next week.
  - HW9 was going to be short anyways, so it should not add a huge burden on your time.
  - HW8 is already posted of course, so if you finish the midterm project up early you can certainly start (or continue) working on it.

Scripts

- There are 50+ learning objectives on the study guide. Your scripts combined should check off at least 20 different objectives.
- You can maybe achieve all that in one script, which is fine, but aim for two. No more than two scripts though, you should be able to fit everything into 2 scripts without difficulty.
- Your scripts can solve problems similar to those we've looked at already this semester, but the problem they are solving should be a new one.
- You are welcome to use the internet, your book, etc for help, but your work should be your own, and you are to work independently.

Explanation Writeups

- For each script, include a separate short write-up about that script. The write-up should include:
  - A description of what problem the script was attempting to solve
  - A list of what learning objectives the script fulfills
  - For each learning objective, include the line number(s) where that objective is first fulfilled

Review Question

The code block to the right starts defining a class. Only 1 of the below options for defining the `increment` method will work. Which one?

```
class BestCounter:
    def __init__(self, start):
        self.counter = start
```

A)
```
def increment(self,value):
    counter += value
```

B)
```
def increment(value):
    counter += self.value
```

C)
```
def increment(self,value):
    self.counter += self.value
```

D)
```
def increment(self,value):
    self.counter += value
```

Accessing and Using Methods

- After definition, you have two main ways to access and use the method
- Dot Notation (Conventional):

```
c = Coordinate(3,4)
O = Coordinate(0,0)
print(c.distance(O))
```

- Function Notation:

```
c = Coordinate(3,4)
O = Coordinate(0,0)
print(Coordinate.distance(c,O))
```

Representation

- Printing out an object that you just created as an instance of a class will look ugly

```
>>> c = Coordinate(3,4)
>>> print(c)
<__main__.Coordinate object at 0x7f942ba13550>}
```

- Can provide methods to teach the interpreter how your object should be represented or displayed when printed
  - Special methods, so have double underscores before and after
    - `__str__`: Informal string representation
    - `__repr__`: Formal string representation

A question of formality

- Formal String representation
  - Commonly used in debugging
  - Needs to contain all the information about the class in unambigous way
  - "What information would I need to exactly replicate this object?"
- Informal String representation
  - What is printed or converted to with `str()`
  - Goal is to be easily readable by humans
  - If not defined, `print` will fall back on using `repr()`

Emulating buildin functions

- When I add two strings together, they really get concatenated.
  - Why?
- For any defined type (class), you can specify or "override" how Python's default functions interact with objects of that type
  - Basically any math or boolean operation can be specified
  - All use the leading and following double underscores
  - Reverse versions of many exist
- Examples:
  - `A + B == A.__add__(B)`
  - `A * B == A.__mul__(B)`
  - `B * A == A.__rmul__(B)`

Special Methods

Multiplication	`A * B`	`__mul__`
Addition	`A + B`	`__add__`
Subtraction	`A - B`	`__sub__`
Float Division	`A / B`	`__truediv__`
Int Division	`A // B`	`__floordiv__`
Raise to power	`A ** B`	`__pow__`
Equals	`A == B`	`__eq__`
Not equals	`A != B`	`__ne__`

More exist and can be found here.