CIS192 Python Programming

The End

Harry Smith

University of Pennsylvania

April 19, 2018
Good Design in Python

- ABCs of Python
- Singleton
- Observer
- Memento
- CIS 192 in Review
- Next Steps
The abc module

- Stands for the "Abstract Base Class" module
- Useful for giving a template for how some object should behave
- Requires some methods to be implemented separately, some come pre-packaged
Defining an ABC

Use the ABCMeta class from abc

```python
from abc import ABCMeta
class MyClass(metaclass=ABCMeta):
    ...
```

Harry Smith (University of Pennsylvania)  
CIS 192  
April 19, 2018
Specifying abstract methods

Use the ABCMeta class from abc

```python
import abc
class MyClass(metaclass=abc.ABCMeta):
    @abc.abstractmethod
def task():
    raise NotImplementedError()
```
Can include partially complete abstract methods

```python
import abc
class MyClass(metaclass=abc.ABCMeta):
    @abc.abstractmethod
def partial_task(x):
    x.sort()
```
...or fully implemented methods

```python
import abc

class MyClass(metaclass=abc.ABCMeta):
    def complete_task(x):
        print(x)
```
Subclassing (registering)

Use the register method gained from the ABCMeta

```python
import abc
from abc_base import MyClass

@MyClass.register
class SubClass():
    def task():
        print("Completed task!")

    def partial_task(x):
        super().partial_task(x)
        print(x[0])

s = SubClass()
>>> s.task()
"Completed task!"
>>> s.complete_task("hi")
```
Singleton Pattern

- What if I need exactly one instance of something?
- How can I guarantee that I’m always getting the one thing?
- Why is this important?
When to Use Singleton Pattern

- File I/O
- Multithreaded operations
- Distributed Computing
Singleton: Main Idea

- Define an inner class for a given object
- Let the inner class handle the behavior
- Let the outer class check if the inner class has already been created
Observer Pattern

- What if I have a many-to-one relation of objects?
- How can I guarantee that each of the many objects are kept up to date?
When to Use Observer Pattern

- Controlling sub-processes or other instances
- Syncing up multiple instances / users
- Displaying server status
Observer: Main Idea

- Let the "one" object inherit from a Notifier class
- Let the "many" classes inherit from an Observer class
- On association, let the notifier store a pointer to observer
- On system update, let the notifier send a message to each observer
Memento Pattern

- What happens when I might enter an illegal state in my program?
- How can I procedurally store, observe, and revert to program states?
When to Use Memento Pattern

- When the legality of some operation cannot be thoroughly checked before execution
- When race conditions are observable but unavoidable
- To programmatically keep track of state
Memento: Main Idea

- Write a "memento" class that is a simple wrapper around some particular state
- Extend the vulnerable/interesting class to have diary and backup methods
- Before risky operations, or at predetermined periods, write a memento object
- When a problem is encountered, revert the class using state stored in Memento.
Python Basics & Fundamentals
Functional Programming
Object-Oriented Programming
Iterators, Generators, Exceptions & IO
Regular Expressions & Other Modules
HTTP Requests / HTML Parsing
Data Analysis
Machine Learning
Natural Language Processing
Web Apps
Artificial Intelligence
Probability & Simulations
If there’s anything I want you to remember...

- Generators! Use this space-efficient tool in place of iterators.
- Lambdas & Functional Programming: they look scary but are incredibly useful
  - Comparators
  - Elements of good design: strategy design pattern
- Make your own classes!
- Using "with open ... as f:" to safely open files!
- List/Set/Generator Comprehensions
- Dictionaries
- Use pip, installing binaries, and taking advantage of libraries
- The power of Python function headers (default kwargs, arbitrarily many positional args).
- Testing code is easy
- Use Decorators to avoid repeating yourself
- Python is excellent for scripting. It’s very useful for solving simple math/probability problems.
- Python makes coding fun.
Scratching the Surface

- Each special topic has MUCH more depth than what we’ve covered this semester.
- Many topics we haven’t mentioned
  - Parallel & Distributed Computing
  - Concurrency
  - Graphical User Interfaces
  - etc.
- Check the website for access to some of these.
What’s Next?

- I hope you find the skills you’ve acquired from CIS 192 useful!
- Build your own side-projects, big or small!
- Learn more about Python!
  - PyCon conference recordings
  - Obey the Testing Goat (TDD in Python + Web Dev)
  - New Coder: practical tutorials in Python
- Join the Python community!
- /r/Python subreddit
- Trending GitHub Python repositories
- StackOverflow Python questions
What *Isn’t* Next?

- Build a game!
  - PyGame book
  - More rudimentary game advice
Manage a database!
▶ Advice on database management
What *Isn’t* Next?

- Learn Django!
  - Django Tutorial
  - Mozilla’s Guides (very nice)
What *Isn’t* Next?

- Deploy Applications!
  - Fabric (for SSH/sysadmin)
  - Launch Flask Apps on AWS
  - Launch Django Apps on AWS
What *Isn’t* Next?

- Build an interpreter
- Write your own programming language
- Build a theorem prover
- Automate boring computer tasks
- Scrape the web
- Do beautiful data analysis
- Generate other code automatically
- Handle 3D animation and modeling in Maya
- Send emails automatically
- Quickly create UI
- Build maps and GIS
- Control traffic
- Simulate drug action