Course Takeaways

Physics91SI Spring 2013

Part 0: The Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Part 1: What to Do When Python Complains at You

Programming in Python is often rewarding, but everyone makes mistakes, and when you make mistakes you make Python unhappy. When Python is unhappy it lets you know. But don’t give up! Python may be verbose, but at least it’s good at communicating what’s going wrong. If you pay close attention, you should find it straightforward to resolve most bugs.

The Zen: Remain calm. Take three slow, deep breaths and remind yourself that everything will be okay.
Listening skills: Python is a *friendly* language: it reminds you gently what you screwed up and where. Look carefully at its error message so that you can get an idea for what’s going wrong. The **traceback** will tell you where it went wrong (which may be different from where you screwed up).
Being one with the bug: If the bug is confusing, try following the code’s execution with **PDB**:

```
	python -m pdb <scriptname>.py
```

For speed, set a breakpoint just before the buggy line and continue until the breakpoint.
Learn from the masters: If the solution isn't obvious, don't hesitate to use **Google**. Do this until it's obvious what the problem is and how you can fix it. Make sure to take breaks or work on other sections, since fresh eyes make any bug seem more solvable.
Search your soul: Devise a solution. Often you'll have to choose between a quick and dirty solution and a longer, shinier one. The former usually works just fine and the latter often takes way longer than expected, so consider choosing the former.
Venture forth: Implement stuff until it works, and then rejoice! As before, take plenty of breaks from working on this problem so that you can approach it with an open mind.

Part 2: Quotes by Which to Code

"While I was working on [devious Python application] I had to do [menial, repetitive task], so I just wrote a Python script. #pythonception" –Tweet from xkcd's Black Hat

`""" "Use builtin functions!" -{} """.format(python_programmer_name)`

"Don't reinvent the wheel!" –Henry Ford, regarding Numpy and SciPy

"Commit, commit, commit! Gosh, I wish I'd committed..." –Anonymous

`["Regexes are %s! <3" % word for word in re.findall(r"Beiber is (\w+)", blog_about_Beiber)]` –Millions of regex and listcomp fans

"I am more of a sponge than an inventor. I absorb ideas from every source." –Thomas Edison, regarding the research and implementation of Python packages

"The weak can never forgive. Forgiveness is the attribute of the strong." –Mahatma Gandhi, regarding the struggle with one of his more belligerent Python bugs

"Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law." –Douglas Hofstadter, author of *Gödel, Escher, Bach*

Part 3: Happy Coding!

Sincerely,
Rex Garland and Gabe Ehrlich,
with course advisors Jason Chaves, Julian Kates-Harbeck, and Ian Tenney