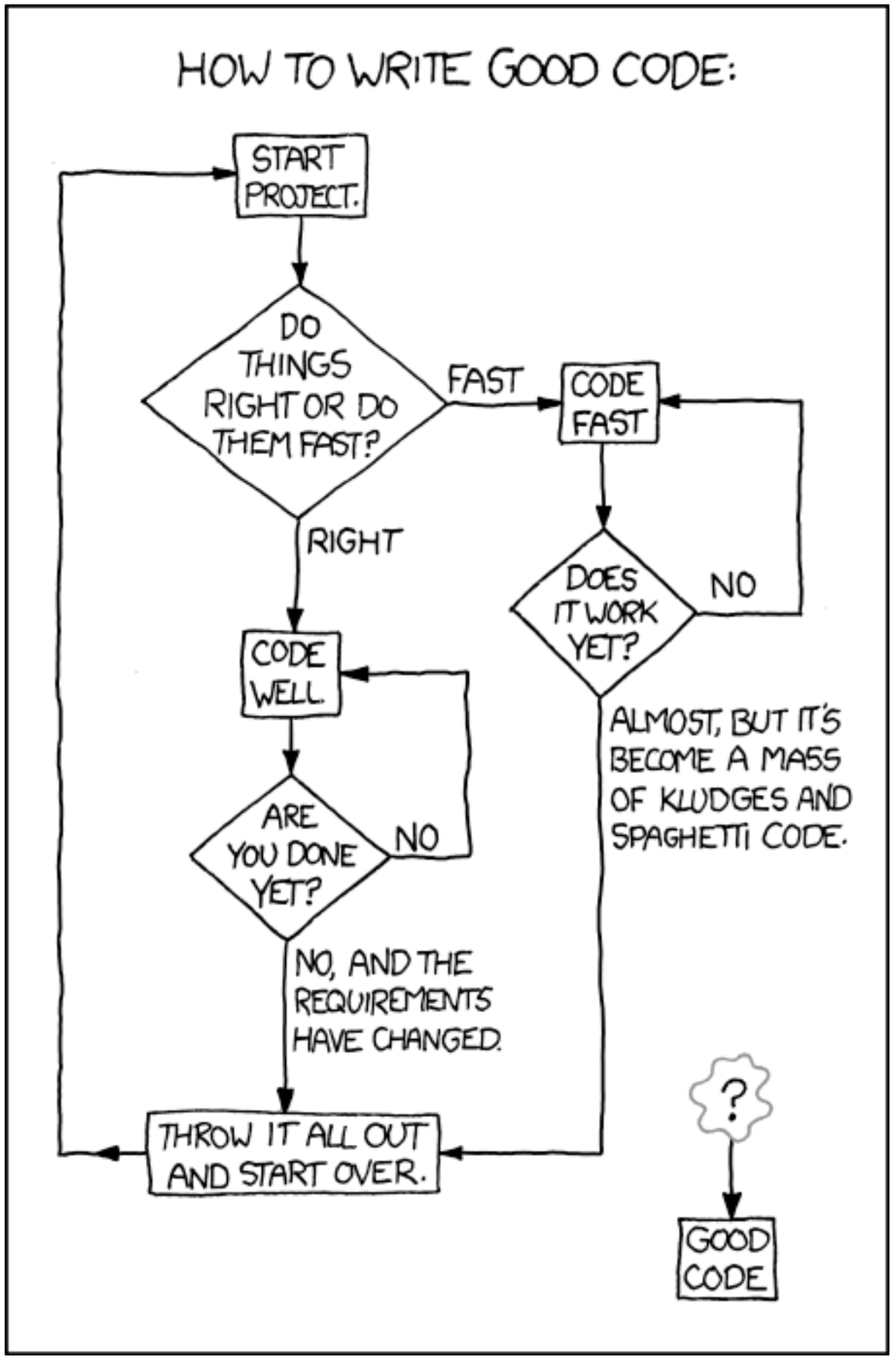
GSPS Productivity Series
Coding Better

Chris Klein

February 15th, 2013


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
RIGHT
DOES IT WORK YET?
NO
CODE WELL.
ARE YOU DONE YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

Topics

1. Text editor selection and personalization
2. Pseudo code
3. Hygiene
4. # Comments ”””
5. Aesthetics
6. Runtime optimization
7. Sharing with collaborators and users
8. Eliminating redundancy

Potpourri of Text Editors

- TextMate, Emacs, Aquamacs, gedit, Notepad++, BBEdit, TextWrangler
  - Or, consider an IDE for your language


Notepad++


gedit

Potpourri of Text Editors

- TextMate, Emacs, Aquamacs, gedit, Notepad++, BBEdit, TextWrangler
  - Or, consider an IDE for your language
  - Edit the preferences! (tabs ➛ spaces, UTF-8)
  - Language specific color coding is awesome
  - Autocompletion of parenthesis groups is awesome
  - Learn keyboard shortcuts (indentation, commenting)
  - Find/replace feature with regex parsing

example.py

Currently Open Documents
- good_programming.txt
- example.py

Last Saved: 2/12/13 3:04:07 PM
File Path ▾ : ~/Desktop/GSPS - Good Coding/example.py

example.py | (no symbol selected)

```
1  """
2  example.py demonstrates good coding habits. This header comment documents the
3  explanation of the program, authorship, creation date, and usage instructions.
4  Author: Christopher Klein   Contact: cklein@astro.berkeley.edu  Date: 2013-02-12
5
6  In the description make sure to include non-standard dependancies, and an
7  explanation of usage including optional arguments and examples. Also, may wish
8  to include the GNU General Public License.
9  """
10 #-----------------------------------------------------------------------------
11 # DECLARATION OF IMPORTS
12 """ Import only what you need. """
13 from scipy import array, median, std, clip, random, append
14
15 #-----------------------------------------------------------------------------
16 # FUNCTION DEFINITIONS
17 def mad_cliping(input_data):
18     """
19     Median Absolute Deviation clipping for input list of numbers. Returns the
20     clipped data, the new median, and the new standard deviation.
21     """
22     medval = median(input_data)
23     sigma = 1.48 * median(abs(medval - input_data))
24     high_sigma_clip_limit = medval + 1 * sigma
25     low_sigma_clip_limit = medval - 1 * sigma
26     clipped_data = input_data.clip(min=low_sigma_clip_limit,
27         max=high_sigma_clip_limit)
28     new_medval = median(clipped_data)
29     new_sigma = 1.48 * median(abs(medval - clipped_data))
30     return clipped_data, new_medval, new_sigma
31
32 #-----------------------------------------------------------------------------
33 # BEGIN MAIN PROGRAM
34 # Create 1000 random numbers (mean=100, sigma=10) and append a 0 and a 1000.
35 random_data = append(random.normal(100, 10, 1000), array([0, 1000]))
36 print median(random_data), std(random_data)
37     # ~100 and ~30
38 # Run the random data through the mad_clipping function to demonstrate results
39 clipped_data, new_medval, new_sigma = mad_cliping(random_data)
40 print new_medval, new_sigma
41     # ~100 and ~10
42
```

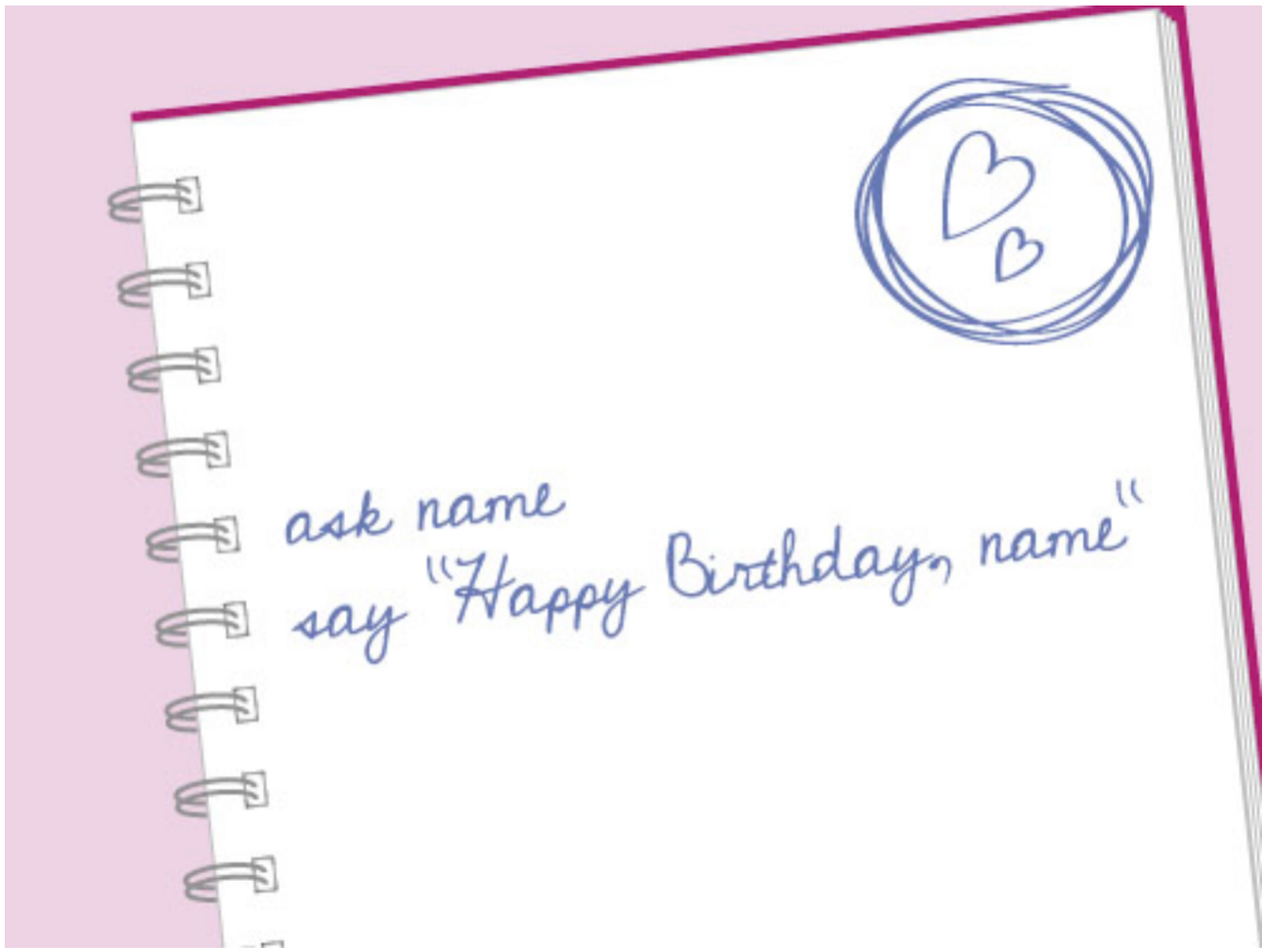
Recent Documents

42 | 5 | Python | Unicode (UTF-8) | Unix (LF) | 1,862 / 196 / 42

Pseudo Code First!!!

- Pseudo code is the logical flow of your program written out in abbreviated English
  - Form a coherent plan before you jump in

Pseudo Code Example




```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
 string name;
 cout << "What's your name? ";
 getline (cin, name);
 cout << "Happy Birthday" << name << "!\n";
 return 0;
}
```

Pseudo Code First!!!

- Pseudo code is the logical flow of your program written out in abbreviated English
  - Form a coherent plan before you jump in
  - Run your pseudo code by colleagues to get high level comments and feedback
    - Often very useful in avoiding roadblocks or incorporating better solutions
    - This is likely the deepest level your advisor will ever see

Hygienic Code

- Try to keep your first draft concise and logical
- Inspect periodically and clean:
  - Remove accumulated hard-coded numbers, outdated comments, and misleading function names
- Do not ignore compiler and runtime warnings
  - Track them down and fix your code to comply

Comment your code!

- Document your code as if someone else might have to take it over at any moment
- Add long header comment at top of file
  - Authorship, creation date, description, usage
- Maintain a README file
- Comment changes on multi-author projects
- Explain the purpose of algorithms
- Avoid pointless comments
- Annotate as your write, don't put it off

example.py

Currently Open Documents
- good_programming.txt
- example.py

Last Saved: 2/12/13 3:04:07 PM
File Path ▾ : ~/Desktop/GSPS - Good Coding/example.py

example.py | (no symbol selected)

```
"""
example.py demonstrates good coding habits. This header comment documents the
explanation of the program, authorship, creation date, and usage instructions.
Author: Christopher Klein   Contact: cklein@astro.berkeley.edu  Date: 2013-02-12

In the description make sure to include non-standard dependancies, and an
explanation of usage including optional arguments and examples. Also, may wish
to include the GNU General Public License.
"""
#-------------------------------------------------------------------------------
# DECLARATION OF IMPORTS
""" Import only what you need. """
from scipy import array, median, std, clip, random, append

#-------------------------------------------------------------------------------
# FUNCTION DEFINITIONS
def mad_cliping(input_data):
    """
    Median Absolute Deviation clipping for input list of numbers. Returns the
    clipped data, the new median, and the new standard deviation.
    """
    medval = median(input_data)
    sigma = 1.48 * median(abs(medval - input_data))
    high_sigma_clip_limit = medval + 1 * sigma
    low_sigma_clip_limit = medval - 1 * sigma
    clipped_data = input_data.clip(min=low_sigma_clip_limit,
        max=high_sigma_clip_limit)
    new_medval = median(clipped_data)
    new_sigma = 1.48 * median(abs(medval - clipped_data))
    return clipped_data, new_medval, new_sigma

#-------------------------------------------------------------------------------
# BEGIN MAIN PROGRAM
# Create 1000 random numbers (mean=100, sigma=10) and append a 0 and a 1000.
random_data = append(random.normal(100, 10, 1000), array([0, 1000]))
print median(random_data), std(random_data)
    # ~100 and ~30
# Run the random data through the mad_clipping function to demonstrate results
clipped_data, new_medval, new_sigma = mad_cliping(random_data)
print new_medval, new_sigma
    # ~100 and ~10

```

Recent Documents

42 | 5 | Python | Unicode (UTF-8) | Unix (LF) | 1,862 / 196 / 42

Documentation Generation

- Consider using automated documentation formatting and generation programs
  - For Python there is Epydoc and Sphinx
  - Doxygen supports nearly any language you would care about

In [1]: plot?

GSPS - Good Coding — less — 98×36

```
Base Class: <type 'function'>
String Form:<function plot at 0x6c2e770>
Namespace:  Interactive
File:       /Library/Frameworks/Python.framework/Versions/7.2/lib/python2.7/site-packages/matplotl
ib/pyplot.py
Definition: plot(*args, **kwargs)
Docstring:
Plot lines and/or markers to the
:class:`~matplotlib.axes.Axes`.  *args* is a variable length
argument, allowing for multiple *x*, *y* pairs with an
optional format string.  For example, each of the following is
legal::

    plot(x, y)         # plot x and y using default line style and color
    plot(x, y, 'bo')   # plot x and y using blue circle markers
    plot(y)            # plot y using x as index array 0..N-1
    plot(y, 'r+')      # ditto, but with red plusses

If *x* and/or *y* is 2-dimensional, then the corresponding columns
will be plotted.

An arbitrary number of *x*, *y*, *fmt* groups can be
specified, as in::

    a.plot(x1, y1, 'g^', x2, y2, 'g-')

Return value is a list of lines that were added.

The following format string characters are accepted to control
the line style or marker:

================    ================================
character           description
================    ================================
``'-'``             solid line style
:
```

plot(**args*, ***kwargs*)

Plot lines and/or markers to the `Axes`. *args* is a variable length argument, allowing for multiple *x*, *y* pairs with an optional format string. For example, each of the following is legal:

```
plot(x, y)         # plot x and y using default line style and color
plot(x, y, 'bo')   # plot x and y using blue circle markers
plot(y)            # plot y using x as index array 0..N-1
plot(y, 'r+')      # ditto, but with red plusses
```

If *x* and/or *y* is 2-dimensional, then the corresponding columns will be plotted.

An arbitrary number of *x*, *y*, *fmt* groups can be specified, as in:

```
a.plot(x1, y1, 'g^', x2, y2, 'g-')
```

Return value is a list of lines that were added.

By default, each line is assigned a different color specified by a 'color cycle'. To change this behavior, you can edit the axes.color_cycle rcParam. Alternatively, you can use `set_default_color_cycle()`.

The following format string characters are accepted to control the line style or marker:

character	description
`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style
`'.'`	point marker
`','`	pixel marker
`'o'`	circle marker
`'v'`	triangle_down marker
`'^'`	triangle_up marker
`'<'`	triangle_left marker

Documentation Generation

- Consider using automated documentation formatting and generation programs
  - For Python there is Epydoc and Sphinx
  - Doxygen supports nearly any language you would care about
- Likely need to modify commenting style to best support the formatting generator
  - But, this will also likely improve your commenting style

Aesthetics

- Standardized style improves readability
  - When/where to skip lines
  - Where to write comments
  - Function and variable naming conventions
  - Whitespace usage
  - Horizontal char limit (80 is standard)
- Very useful for multi-author projects
- Do not rewrite code just to alter the style
  - This wastes time and could change behavior

Python PEP 8

PEP 8 -- Style Guide for Python Code

www.python.org/dev/peps/pep-0008/

Reader

python™

search

Advanced Search

» PEP Index > PEP 8 -- Style Guide for Python Code

ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
下载
COMMUNITY
FOUNDATION
CORE DEVELOPMENT

Python Wiki
Python Insider Blog
Python 2 or 3?
Help Fund Python

Non-English Resources

Release Schedule

Saturday, February 16
2.7.4 final
3.2.4 final
Saturday, March 2
3.3.1 final

PEP:	8
Title:	Style Guide for Python Code
Version:	89db18c77152
Last-Modified:	2013-01-13 11:28:10 +0100 (Sun, 13 Jan 2013)
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>
Status:	Active
Type:	Process
Content-Type:	text/x-rst
Created:	05-Jul-2001
Post-History:	05-Jul-2001

Contents

Display a menu

PEP 8 -- Style Guide for Python Code

www.python.org/dev/peps/pep-0008/

Reader

Showing events until 4/15. Look for more

Google Calendar

Events Calendar

Sunday, February 24
PyCon Russia 2013
Monday, February 25
PyCon Russia 2013
Wednesday, March 13
PyCon US 2013
Thursday, March 14
PyCon US 2013
Friday, March 15
PyCon US 2013

Google Calendar

Add an event to this calendar.

User Group Calendar

Tuesday, February 12	
7:00pm	Leipzig Pyt
Tuesday, February 26	
6:00pm	Python She
Monday, March 4	
7:00am	Melbourne

Introduction

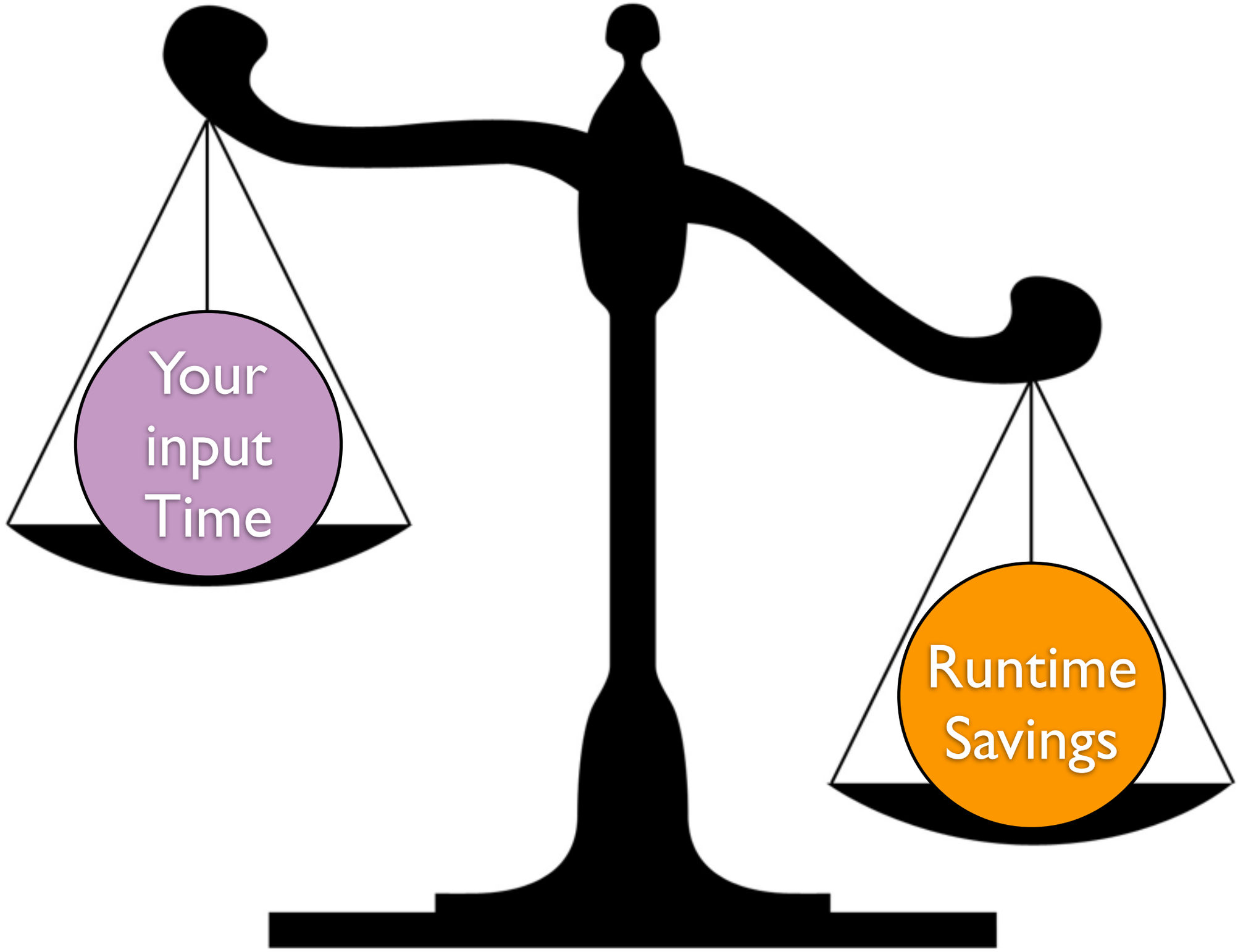
This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python [1].

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

Display a menu

Runtime Optimization

- Assess the runtimes for various components
  - ipython's timeit, or basic print/logging lines
- Always go after the bottleneck first
  - Pareto principle (80% - 20% rule of thumb)
  - Avoid premature optimization
- Consider packaging computational code in lower-level language
- Balance human time investment with computation time savings


Your input Time
Runtime Savings

Before you Share

- Assume your users are stupid and lazy
  - Write robust code, anticipate (user) errors
- Ensure security
  - Never hard-code passwords, avoid OS sys calls
  - Import only what you need
- If GUI, include contextual help and instructions
- Validate all input parameters
  - Ensure malformed user input is harmless
- Provide useful error messages

Write Once, Use Forever

- Throughout career buildup your own commonly-used modules and codebases
- Maintain and import these modules to save yourself from rewriting slightly different versions for each new project
  - Converting coords or date-time (UT, MJD, HJD)
  - Plotting
  - Function fitting (gaussian), basic statistics
  - Parsing text files (i.e., catalog query results)

SciPy Module

- random
- array
- median
- std
- clip
- append

... many more

MyStats Module

- mad_clipping

gaussian_fitting

... many more

Imported into example.py