# This isn’t academia

Writing real code for the real world

BUSINESS
CONSULTANTS

DEEP
TECHNOLOGISTS

# Clean Code – Starting With The End in Mind

**What I want you to take away from this discussion**

1. What is ‘Clean Code’
2. Why is it important
3. Start at the beginning – Know your requirements
4. Patterns and Idioms
5. Pythonic python
6. Python style guide
7. Test driven development (TDD) in Python
8. Neat Jupyter Notebook tricks

## Clean Code – In a simple quote

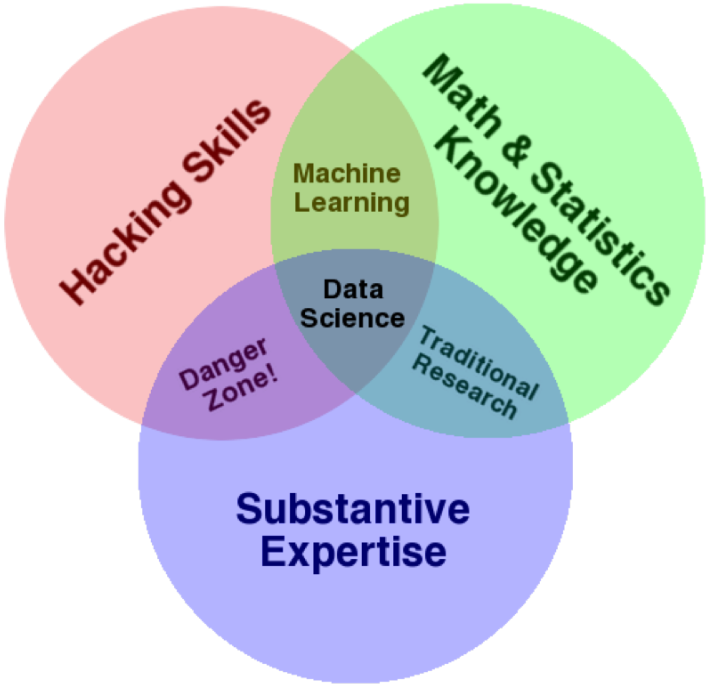
"Programming is an art of telling another human what one wants the computer to do."

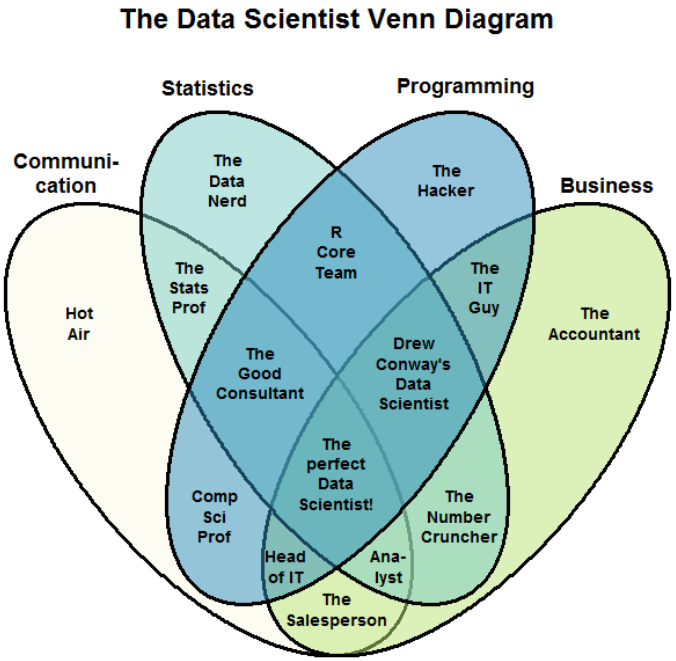
-Donald Knuth

# Clean Code – How to get there

| | |
|---|---|
| Engineering Practices | But, but, but I am a data scientist, not some mere programmer! I am an artiste. No. |
| Pythonic Code | I write code however I want to! I am a special snowflake!. No. |
| Testing | It builds, ship it! No. |
| Documentation | Code is inherently self-documenting. No. |

# Clean Code – Venn diagram madness

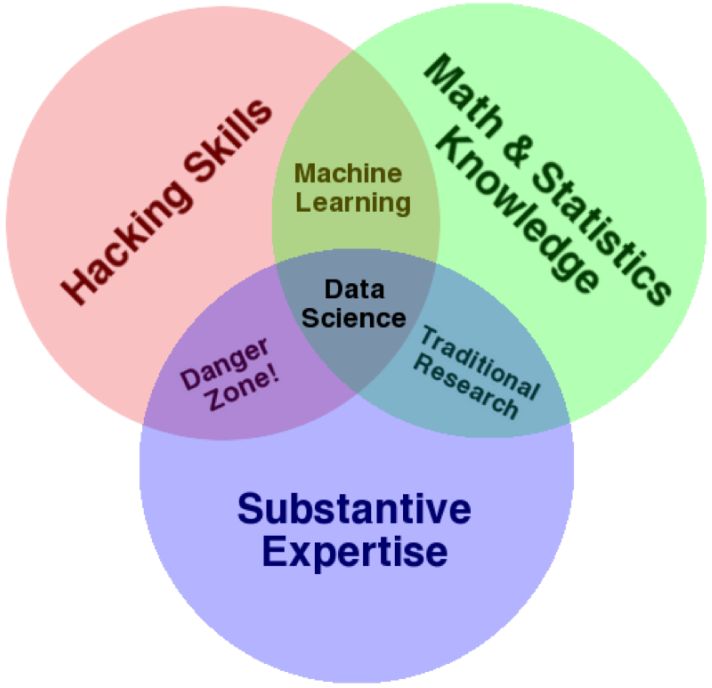


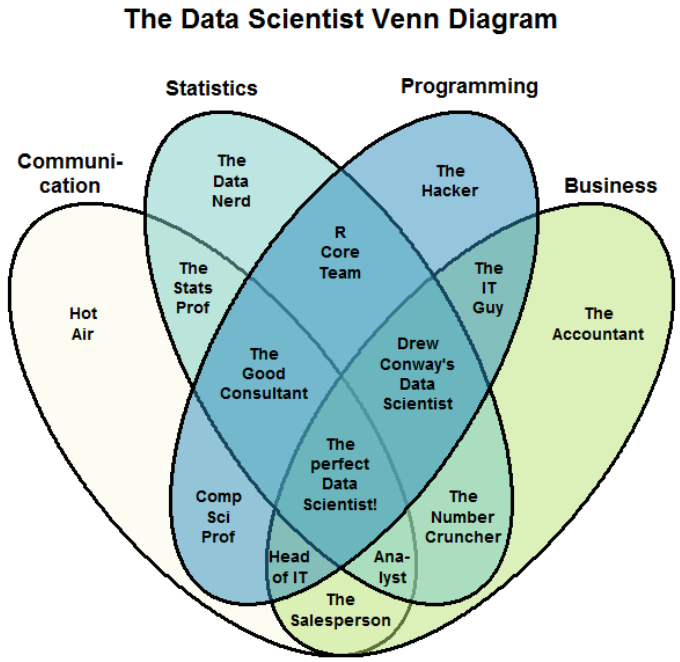

Which is more accurate?

# Clean Code – Venn diagram madness (continued)

Engineering




Engineering




Neither. They are missing the crucial element: engineering

# Clean Code – Engineering in practice

**What is clean code? Why is it important?**

Clean code is code you remember – Configuration management tools have very long memories. Long after you have forgotten about a project, a project still remembers you.

Clean code is code that your team mates can work with – Any good developer should have one goal in mind: code themselves out of a job. If your team mates can pick up your code and run with it, you can find more interesting things to do.

Clean code is malleable – You can update it, extend it, move it to a new project with minimal pain

Clean code is easy to debug – Bugs, sometimes critical bugs, will happen. The difference between an hour fix and a weekend fix will be how clean is the code.

# Clean Code – Engineering in practice (continued)

**What is clean code? Why is it important?**

Clean code is thoroughly tested – Unit tests are cheap insurance against catastrophic failures; if you rely on integration tests, you are doomed.

Clean code is loosely coupled – Code should not be overly dependent on others. This includes being dependent on libraries.

Clean code is small – Classes and functions should only carry as much code as they need.

Clean code is focused – Any aspect of an application should be abstractable to a single requirement.

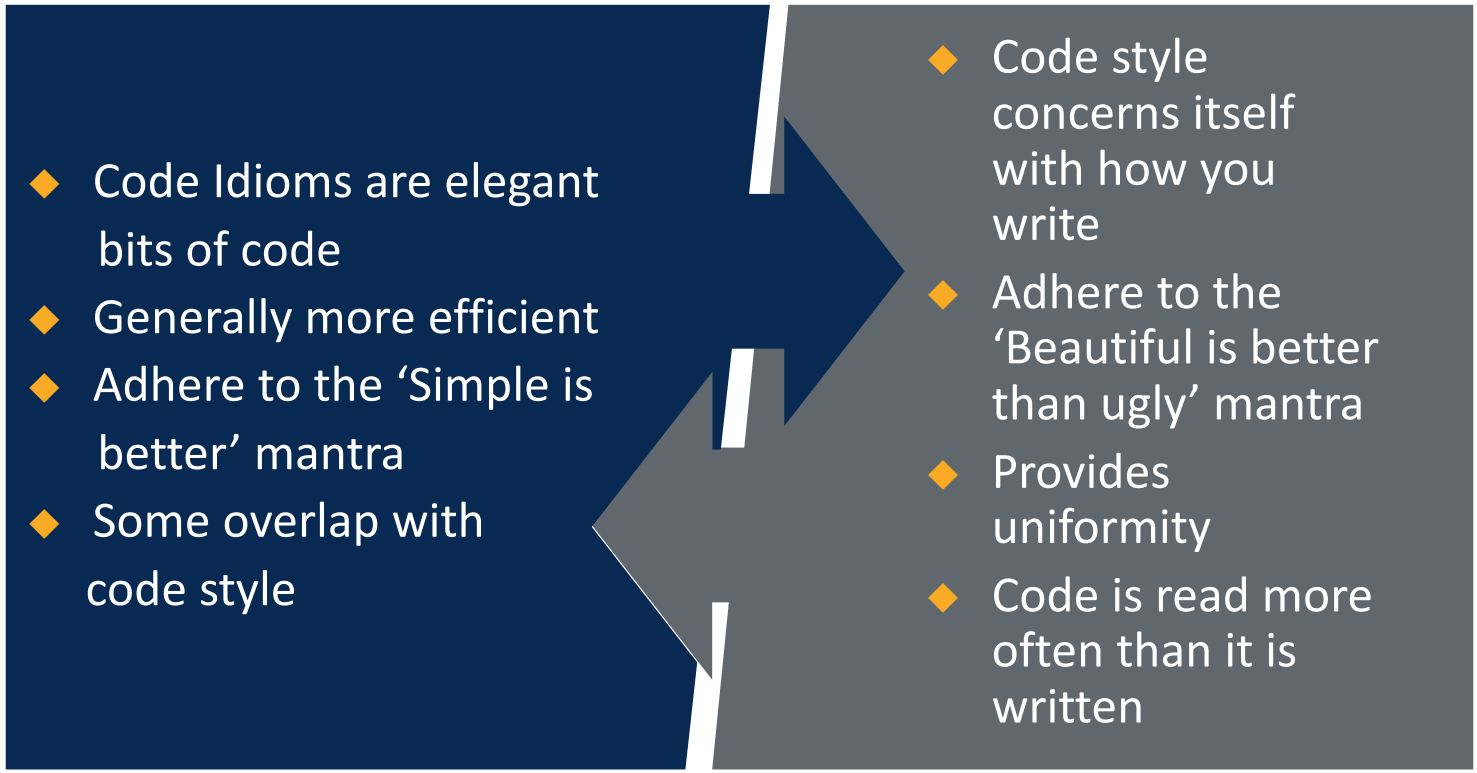
# Clean Code – Understand your requirements

**Seek to Understand**

**Much of the discussion on clean code orbits around syntax: naming conventions, programming paradigms, and patterns. These are all very important, but before you get there, what are you trying to write?**

Know who created the requirement
Know the business use case of the requirement
Know the input of the requirement
Know what provides the input to the requirement
Know the output of the requirement
Know what consumes the output of the requirement

# Pythonic Python – Code idioms vs. Code styles



# Pythonic Python – Python Idioms

## Swap variables

Instead of:

```
temp = a
a = b
b = temp
```

Use:

```
b, a = a, b
```

Sorcery! No, tuples

## Concatenating strings

Instead of:

```
result = ''
for s in colors:
    result += s
```

Use:

```
result = ''.join(colors)
```

## Use ‘in’ to iterate

Instead of:

```
for i in range(len(l)):
    e = l[i]
    use e
```

Use:

```
for e in l:
    use e
```

## Use get(key,default)

Instead of checking for a key in a dictionary then returning a default value, return a predefined default value

## Use setdefault()

Conceptually the same as above, but sets the value from the dictionary to the default value

## Use zip() to make dictionaries

The zip command will take two lists and merge them into a single dictionary

# Pythonic Python – Python Idioms

**Testing for truth**

Instead of:

```
if x == True:
    do something
```

Use:

```
if x:
    do something
```

**Use in/not in**

Instead of using contains or iterating through a list for membership use in or not in:

```
if val in my_list:
if val not in my_list:
```

**Using intrinsic truth**

Instead of:

```
if len(l) != 0:
    do something
```

Use:

```
if l:
    do something
```

**.....**

- .....
- .....
- .....

**Use enumerate for index & value**

The enumerate() returns the index and value from a list:

```
for (index, item) in enumerate(items):
```

**.....**

- .....
- .....
- .....

# Pythonic Python – Python Code Style

## Code layout

- Indent using spaces
- Indent 4 spaces
- Limit all lines to a maximum of 79 characters; 72 for comment blocks

## Blank lines

- Surround top-level function and class definitions with two blank lines
- Method definitions inside a class are surrounded by a single blank line

## Wrapping variable lists

```
def foo(var_1, var_2, …
            var_n
def foo(
   var_1, var_2
```

## Imports

- One import per line
- Unless using ‘from’
- Group by standard library, related third party, then local application/library

## Match operators & operands

```
income = (gross_wages
        + taxable_interest
        + (dividends - qualified_dividends)
        - ira_deduction)
```

## Quoted strings

- Single quote or double quote
- Just stick to a style

# Pythonic Python – Python Code Style

## Avoid extraneous whitespace

- spam( ham[ 1 ], { eggs: 2 } )
- bar = (0, )
- spam  (1)

## Use extraneous whitespace

- i = i + 1
- submitted += 1
- c = (a + b) * (a - b)

## Conditional blocks

- Complex conditionals can be logically separated into new lines
- Avoid placing executed code on the same line as the conditional

## Commenting

- Yes
- Comments should match the code
- When writing English, follow Strunk and White.

## Naming conventions

- Variables and functions: use lowercase with underscores
- Classes use camel case
- Avoid ‘l’, ‘O’, and ‘I’

## Naming conventions (cont)

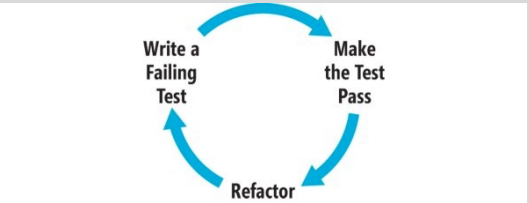
- Constants:use upper care with underscores
- Consistent, meaningful names

# Testing – Test Driven Development

## Philosophy

- What is it? The process of implementing code by writing your tests first
- Why? To test the code, you have to understand what your code is supposed to do
- Think about what your code might do

## Practice




- There are a number of tools available to help with this: ‘nose’, ‘PyUnit’, ‘Pyscope’
- The problem is not the practice, it is the discipline

## Testing Algorithms

- Break your processing down into atomic functions
- Know your data: know your edge cases, median, mean, etc.
- Prove your math using very small data sets

# Clean Code – Useful Tools

| | |
|---|---|
| Unit Test Generator | Pythonscope – Pythoscope is a unit test generator for programs written in Python. It's open source, licensed under the MIT license. |
| Code cleanup | Pylint – Analyzes Python source code looking for bugs and signs of poor quality |
| Duplicate Finder | Clone Digger – Discovers duplicate code in Python |
| Code Metrics | Pymetrics – Produces metrics for Python programs. Metrics include McCabe's Cyclomatic Complexity metric, LoC, %Comments, etc. Users can also define their own metrics using data from PyMetrics. |

# Jupyter Notebooks – Ipython Documentation

## Compatibility

- The Jupyter notebook is an implementation of Ipython
- Check the version in your Jupyter notebook for compatibility

## Code Blocks

- Can put LaTex in code blocks
- Can use raw HTML
- Large number of Python print formatting options

## Markdown Blocks

- Can show LaTex in either inline or stand alone
- Can use raw HTML
- Lots of formatting commands

IPython Documentation

# Jupyter Notebooks – Markdown language basics

| Language | Result |
| --- | --- |

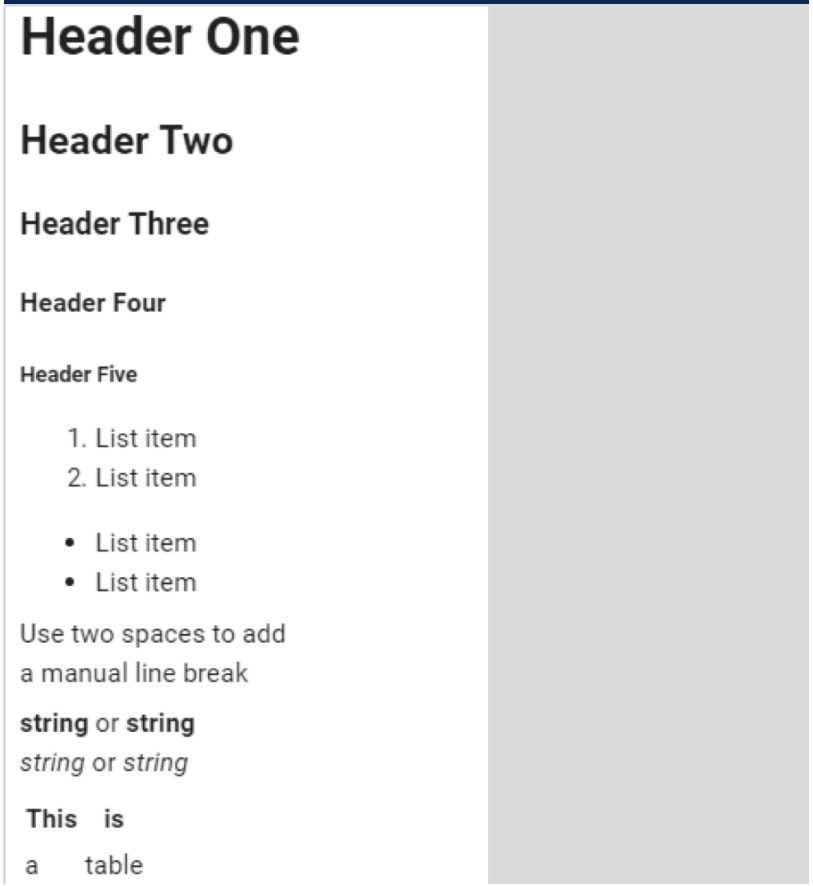
```
# Header One
## Header Two
### Header Three
#### Header Four
##### Header Five
1.  List item
2.  List item


*   List item
*   List item

Use two spaces to add
a manual line break

__string__ or **string**
_string_ or *string*

| This | is   |
|------|------|
|  a   | table|
```



# Jupyter Notebooks – Embedded HTML

## Language

Inside a code block:
from IPython.core.display import HTML
and wrap your html in HTML('')

Inside markdown you can change font text:
<font color=blue>Text</font>

Or build tables

```
<table style="width:100%">
 <tr>
  <th>Firstname</th>
  <th>Lastname</th>
  <th>Age</th>
 </tr>
 <tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
 </tr>
 <tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
 </tr>
```

## Result

link

Text

| Firstname | Lastname | Age |
|---|---|---|
| Jill | Smith | 50 |
| Eve | Jackson | 94 |

# Jupyter Notebooks – LaTex

| Language | Result |
| --- | --- |
| To print within code, import the display, Math, and Latex classes from IPython.display<br>Use as follows:<br>display(Math(r'F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k} dx')) | $F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k} dx$ |
| In markdown, use '$' around the equation for an inline display $e^{i\pi} + 1 = 0$ | $e^{i\pi} + 1 = 0$ |
| In markdown, use '$$' around the equation to set it apart: $$e^x=\sum_{i=0}^\infty \frac{1}{i!}x^i$$ | $e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$ |

# Writing Clean Code - Bibliography

**Good artists copy; great artists steal**


1. Importance of Writing Clean Code
2. Writing clean, testable, high quality code in Python
3. Python For Engineers - Writing Great Code
4. Write Clean, Professional, Maintainable, Quality Code in Python
5. A Quick Primer on Writing Readable Python Code for New Developers
6. Python 3 Patterns, Recipes and Idioms
7. The Hitchhikers Guide to Python
8. Python Style Guide
9. Test Driven Python Development
10. Docs » Examples » Markdown Cells
11. Docs » Examples » Motivating Examples

# Writing Clean Code – Bibliography (continued)

**When in doubt, go to the original source**


12. Seven Habits of Highly Effective People
13. Code Like a Pythonista: Idiomatic Python
14. Picasso, maybe
15. Beginning Test-Driven Development in Python

## Closing

# Questions?

Jim Taber
Senior Analytics Consultant
West Monroe Partners
jtaber@wmp.com