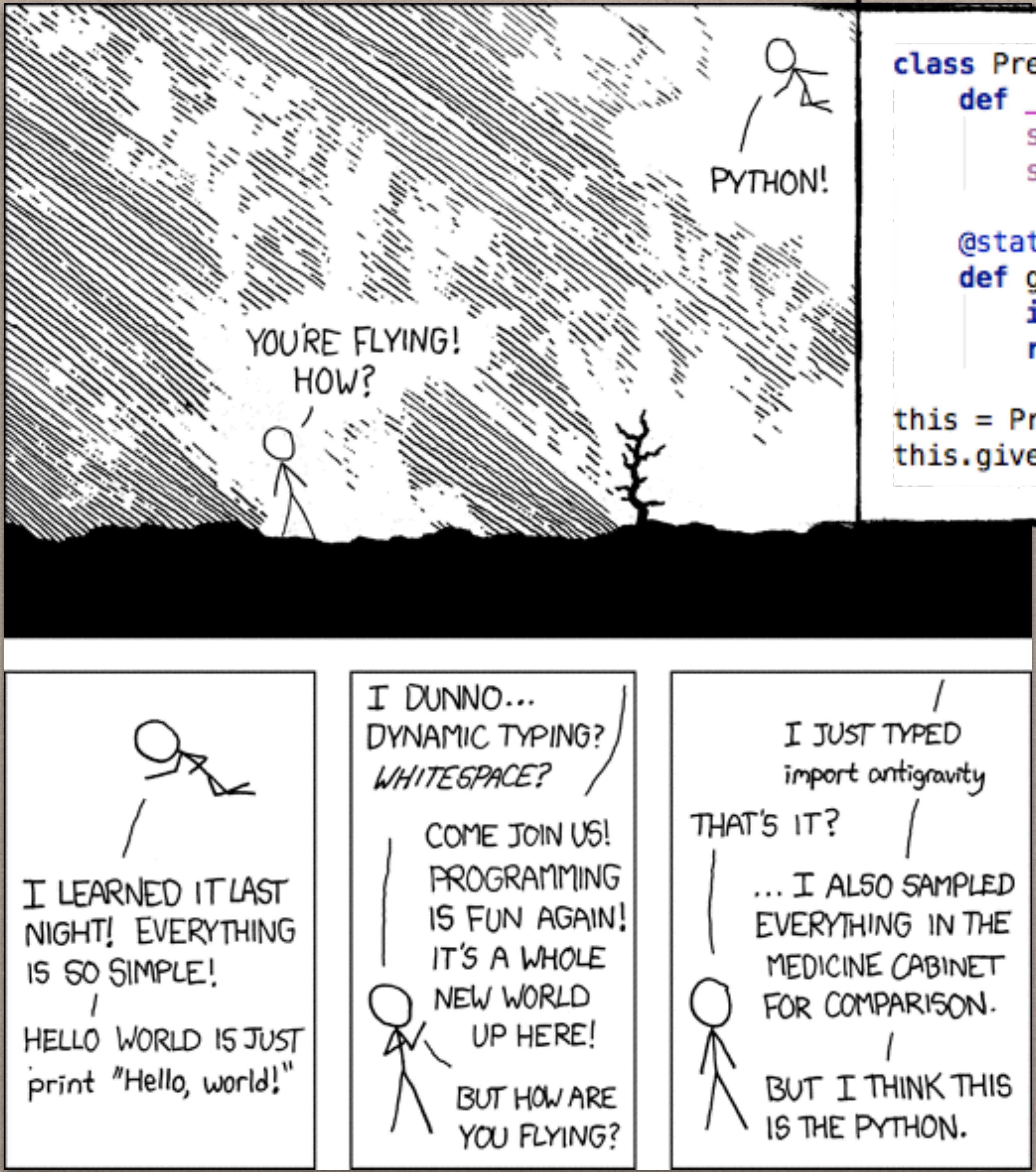


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

```python
class Presentation(object):
    def __init__(self, topic):
        self.topic = topic
        self.presenter = "Dan O'Day"

    @staticmethod
    def gives_you_wings(self):
        import antigravity as wings
        return wings

this = Presentation("Why Python for 4n6?")
this.gives_you_wings()
```

MITRE

ABOUT ME

- Digital forensic examiner, Lake County HIDTA (since 2009)
- Training contractor, viaForensics/NowSecure (since 2012)
- Adjunct instructor: Purdue University Calumet, Governor's State University
- Training includes DHS/FLETC, DOJ, Guidance Software, X-Ways, and more….
- Background: US Army Signal Corps (also an amateur radio operator), criminal intelligence analyst
- M.S. Technology, Purdue University; began Ph.D. but…

OBJECTIVE

- This is not a Python tutorial
- Show a couple of examples illustrating why Python is ~~awesome~~ the *lingua franca* of digital forensics scripting

WHY PYTHON FOR FORENSICS?

- Open source
- Easier to learn
- More with less
- There's a~~n app~~ library for that
- Adds up to increased productivity

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!

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!

python™

Iterate though list/array with iterator/counter integer

"Other" Language

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

int main ()
{
    double myList[5] = {5.0, 2.7, 32.56, 22.1, 7.9};
    myListLength = sizeof(myList) / sizeof(myList[0]);
    for (i=0; i < myListLength; i++)
    {
        cout << "Element " << i << " = " << myList[i];
        cout << endl;
    }
    return 0;
}
```

Iterate though list/array with iterator/counter integer

Python Antipattern

```
my_list = [5.0, 2.7, 32.56, 22.1, 7.9]

i = 0
for item in my_list:
    print ‘Element %d = %0.2f’ % (i, item)
    i += 1
```

Python gives you wings!

```
for i, item in enumerate(my_list):
    print ‘Element %d = %0.2f’ % (i, item)
```

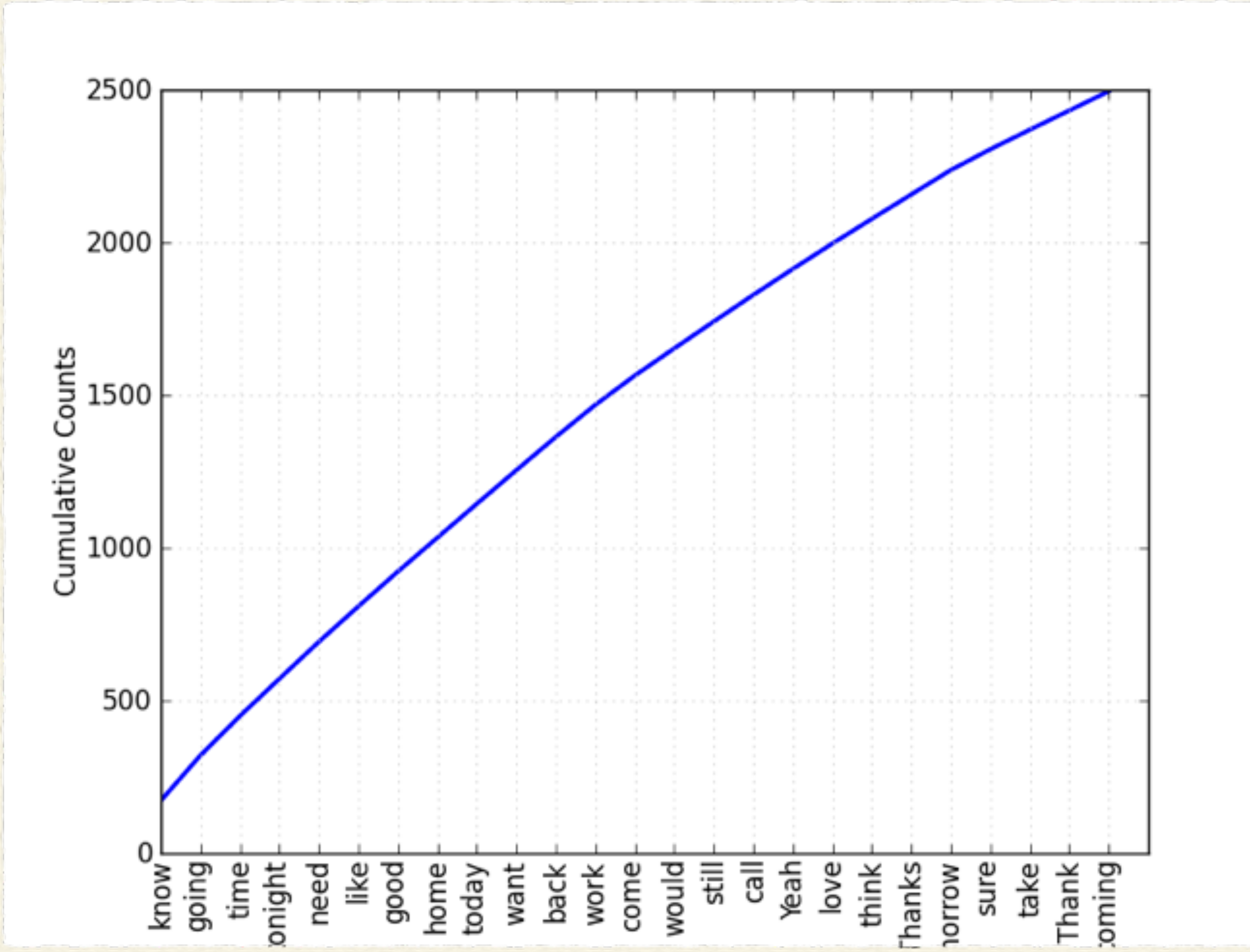
What if we wanted to perform an operation on each list item? Dictionary & list comprehensions

```
my_list = [5.0, 2.7, 32.56, 22.1, 7.9]

my_dict = {n: n**2 for n in my_list}

my_list = [n**2 for n in my_list]
```

Collections

```
from collections import Counter

words = [‘mitre’, ‘ffrdc’, ‘mitre’, ‘research’]

word_count = Counter(words)
print word_count

# output
Counter({'mitre': 2, 'ffrdc': 1, 'research': 1})
```

Collections

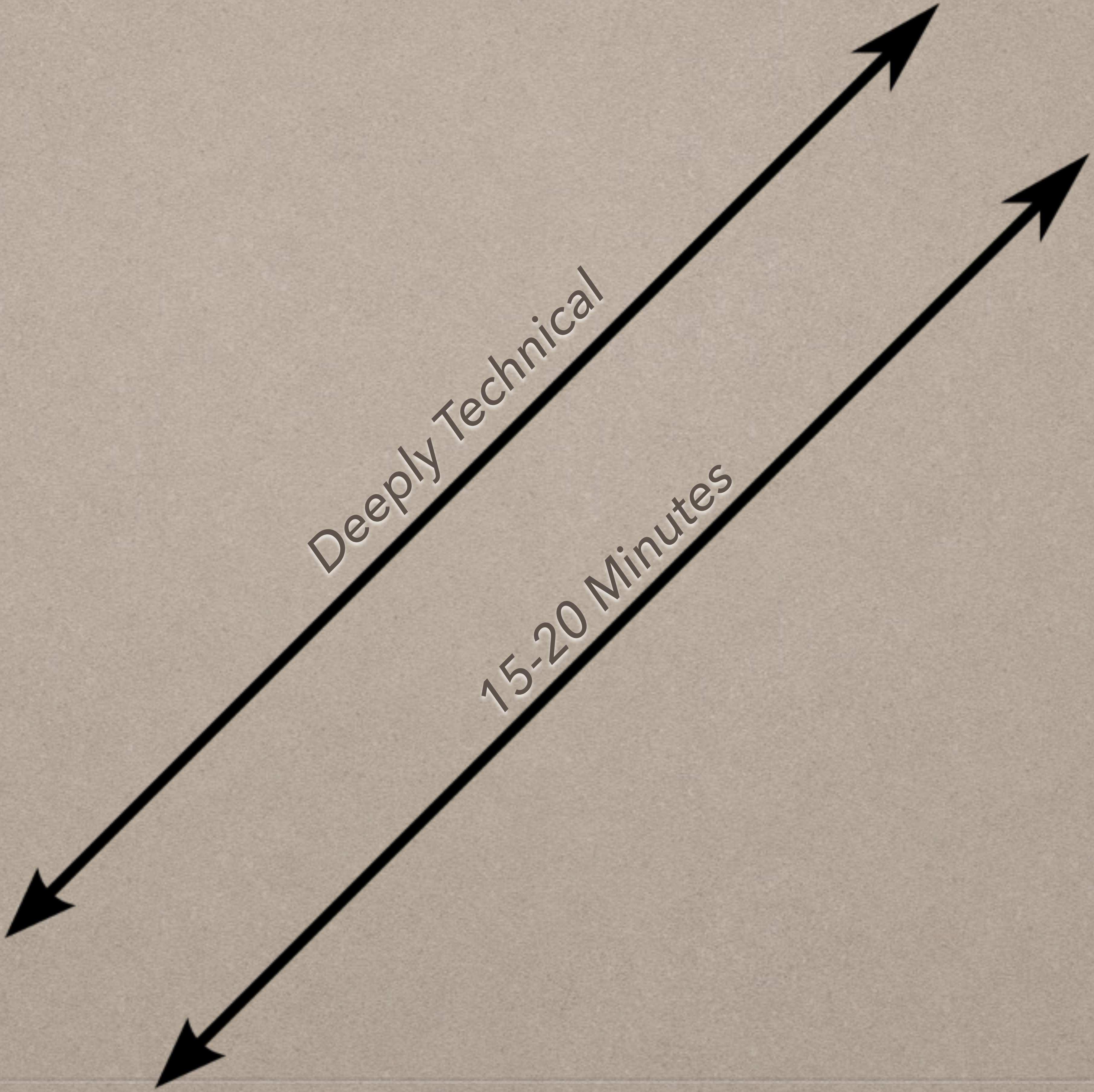


Itertools

```
import binascii, hashlib, itertools

values = [0, 1, 2, 3, 4, 5, 6, 7, 8]
sha1sum = # hash value from gesture.key file
for i in range(3, 10):
    perms = itertools.permutations(values, i)
    for p in perms:
        pattern = ''.join(str(val) for val in p)
        key = binascii.unhexlify(‘'.join(
              '%02x' % (ord(c) - ord('0')) \
              for c in pattern))
        sha1 = hashlib.sha1(key).hexdigest()

        if sha1 == sha1sum:
            return pattern

return None
```

BRIEF MENTION

- Easy to parse data (JSON, CSV, HTML, XML, database, binary - no problem!)
- Generators & Coroutines
- Easy to distribute code as modules and packages
- I want to… go ahead. Python will probably let you


Deeply Technical
15-20 Minutes