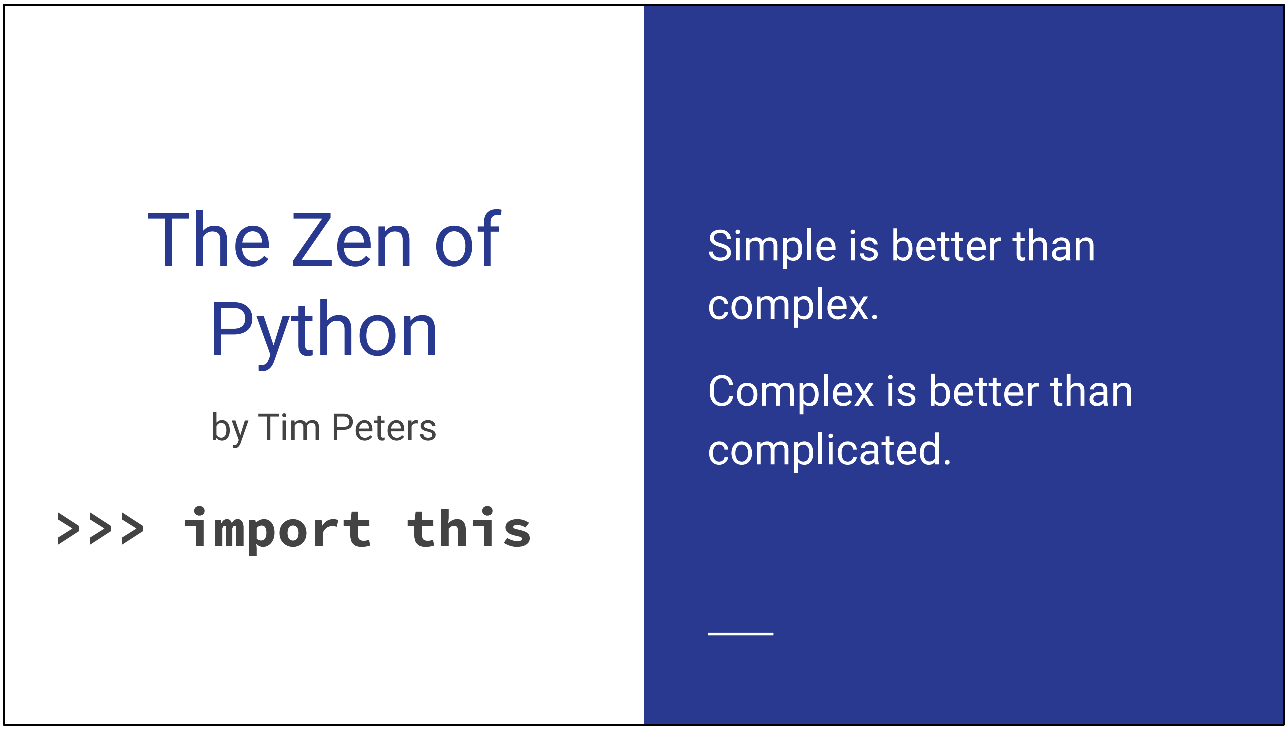
Fight Night in Literate Python

Grant Jenks
June 18, 2019

Let’s get ready to rummmble!!!



The Zen of Python

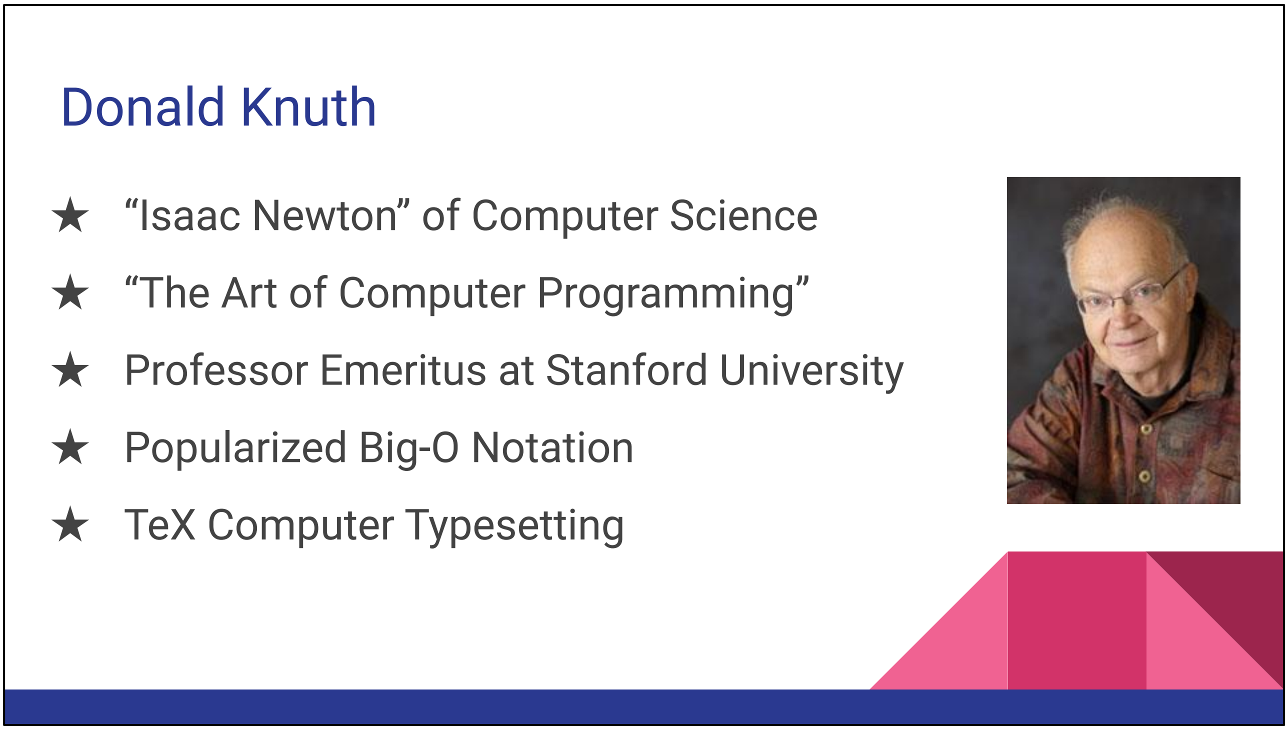
by Tim Peters

```
>>> import this
```

Simple is better than complex.

Complex is better than complicated.

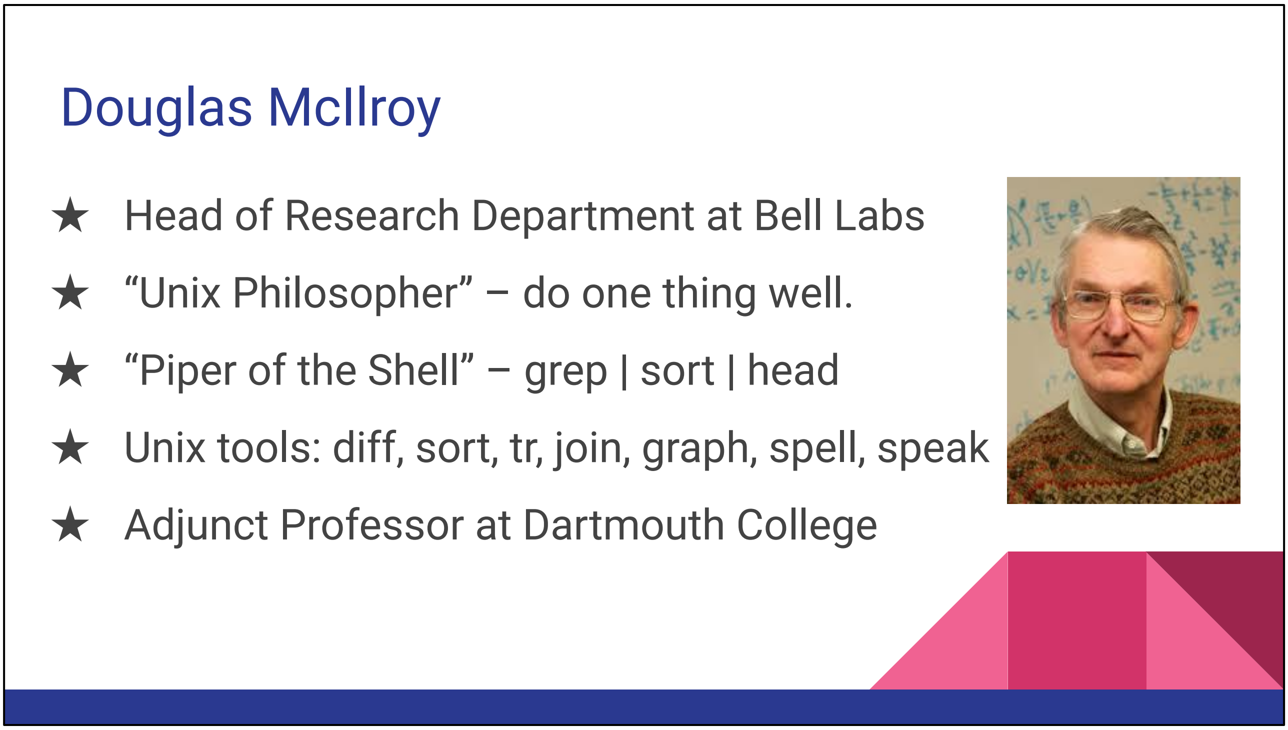
- What’s the difference between complex and complicated?
- Look up in Webster: blah blah blah, see complex; blah blah blah, see complicated.



Donald Knuth

- “Isaac Newton” of Computer Science
- “The Art of Computer Programming”
- Professor Emeritus at Stanford University
- Popularized Big-O Notation
- TeX Computer Typesetting

- Literate Programming



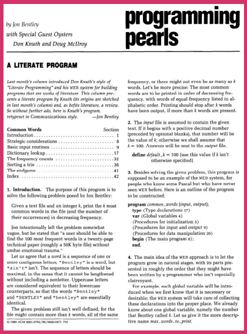
Douglas McIlroy

- ★ Head of Research Department at Bell Labs
- ★ “Unix Philosopher” – do one thing well.
- ★ “Piper of the Shell” – grep | sort | head
- ★ Unix tools: diff, sort, tr, join, graph, spell, speak
- ★ Adjunct Professor at Dartmouth College

- Shell Programming

Challenge

1. Read a file.
2. Parse the words.
3. Tally the frequency.
4. Print the top-10.



by *Jon Bentley*
with Special Guest Oysters
Don Knuth and Doug McIlroy

programming pearls

A LITERATE PROGRAM

Last month's column introduced Don Knuth's style of "Literate Programming" and his WEB *system for building programs that are works of literature. This column presents a literate program by Knuth (its origins are sketched in last month's column) and, as befits literature, a review. So without further ado, here is Knuth's program, retypeset in* Communications *style.* —*Jon Bentley*

1. Introduction. The purpose of this program is to solve the following problem posed by Jon Bentley:

> Given a text file and an integer *k*, print the *k* most common words in the file (and the number of their occurrences) in decreasing frequency.

Jon intentionally left the problem somewhat vague, but he stated that "a user should be able to find the 100 most frequent words in a twenty-page technical paper (roughly a 50K byte file) without undue emotional trauma."

Let us agree that a *word* is a sequence of one or more contiguous letters; "Bentley" is a word, but "ain't" isn't. The sequence of letters should be maximal, in the sense that it cannot be lengthened without including a nonletter. Uppercase letters are considered equivalent to their lowercase counterparts, so that the words "Bentley" and "BENTLEY" and "bentley" are essentially identical.

The given problem still isn't well defined, for the file might contain more than *k* words, all of the same frequency; or there might not even be as many as *k* words. Let's be more precise: The most common words are to be printed in order of decreasing frequency, with words of equal frequency listed in alphabetic order. Printing should stop after *k* words have been output, if more than *k* words are present.

2. The *input* file is assumed to contain the given text. If it begins with a positive decimal number (preceded by optional blanks), that number will be the value of *k*; otherwise we shall assume that *k* = 100. Answers will be sent to the *output* file.

```
define default_k = 100 {use this value if k isn't
         otherwise specified}
```

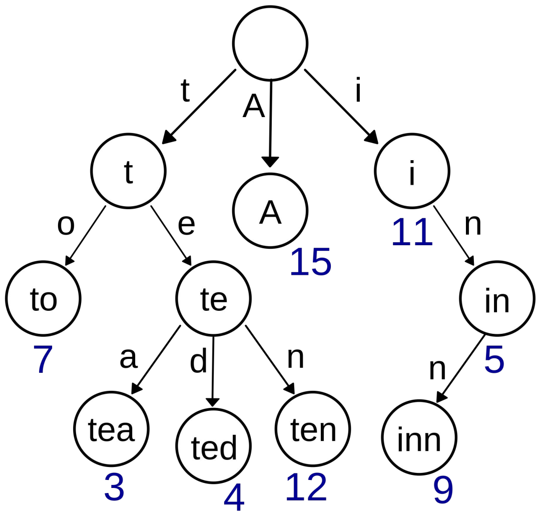
3. Besides solving the given problem, this program is supposed to be an example of the WEB system, for people who know some Pascal but who have never seen WEB before. Here is an outline of the program to be constructed:

```
program common_words (input, output);
  type ⟨Type declarations 17⟩
  var ⟨Global variables 4⟩
  ⟨Procedures for initialization 5⟩
  ⟨Procedures for input and output 9⟩
  ⟨Procedures for data manipulation 20⟩
  begin ⟨The main program 8⟩;
  end.
```

4. The main idea of the WEB approach is to let the program grow in natural stages, with its parts presented in roughly the order that they might have been written by a programmer who isn't especially clairvoyant.

For example, each global variable will be introduced when we first know that it is necessary or desirable; the WEB system will take care of collecting these declarations into the proper place. We already know about one global variable, namely the number that Bentley called *k*. Let us give it the more descriptive name *max_words_to_print*.

Solutions published in Programming Pearls magazine.
Interview-question by today's standards.
Worthy of study in the 1980s.
There's a lot at stake!



Knuth’s Solution

Prefix/Radix-Tree AKA “Trie”
Optimal in Time and Space
“Literate” Programming

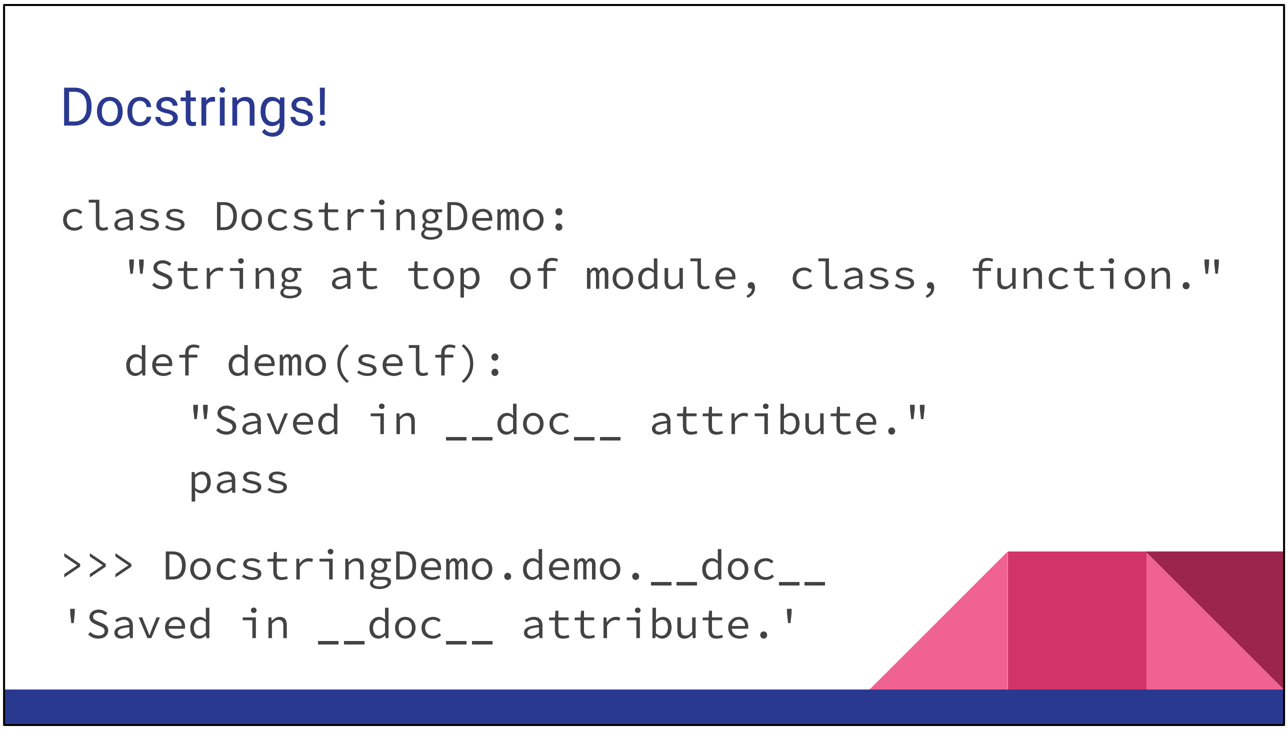
```
1)  tr -cs A-Za-z \n |
2)  tr A-Z a-z |
3)  sort |
4)  uniq -c |
5)  sort -rn |
6)  sed ${1}q
```

McIlroy’s Solution

Multi-Process Support
Larger Than “Memory” Support
“Shell” Programming

Who won?

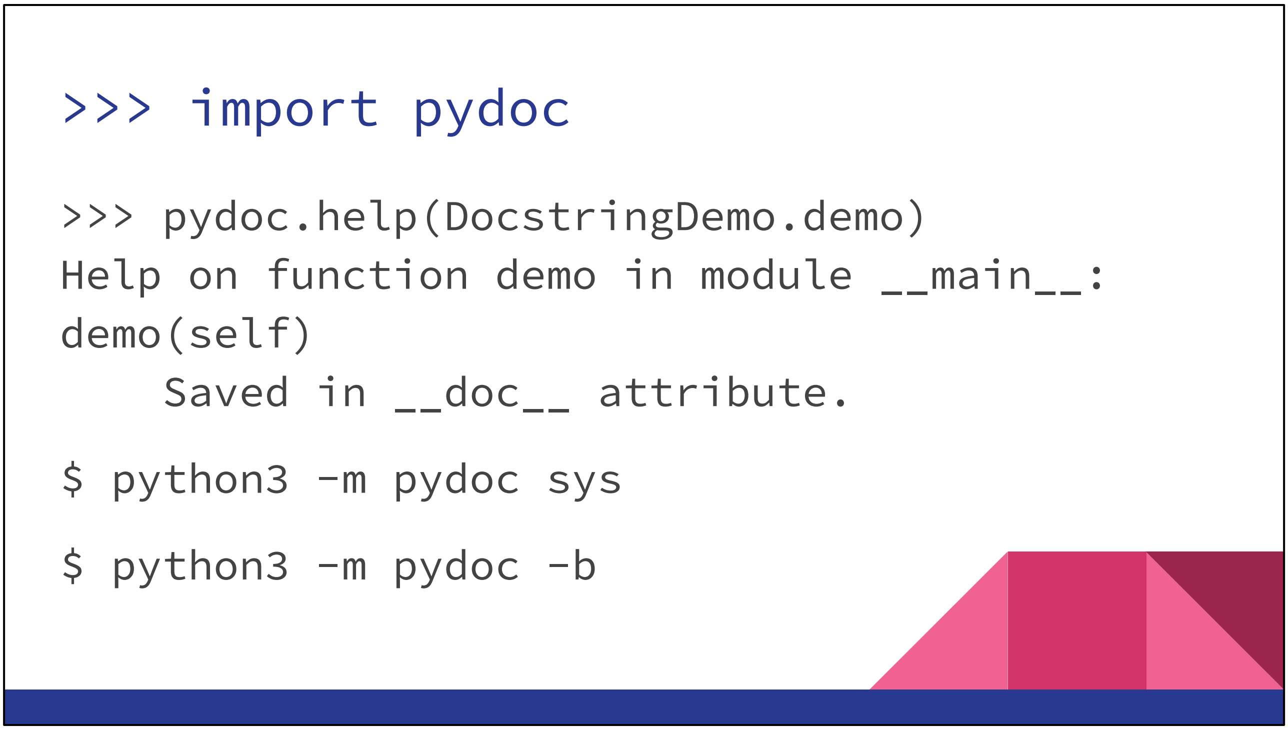
- How many have written a literate program? How many have written a shell program? McIlroy won.
- Which solution is complicated? Which is complex?
- Knuth’s Solution — Complicated, irreducible complexity.
- McIlroy’s Solution — Complex, composition of simple things.
- How does Python fit in the landscape between these extremes: Literate Programming and Shell Programming?



Docstrings!

```
class DocstringDemo:
  "String at top of module, class, function."

  def demo(self):
    "Saved in __doc__ attribute."
    pass

>>> DocstringDemo.demo.__doc__
'Saved in __doc__ attribute.'
```

Such a rare feature of programming languages! So simple and yet so useful.
Elixir, Lisp, and Python have it; everyone else re-purposes comments :(



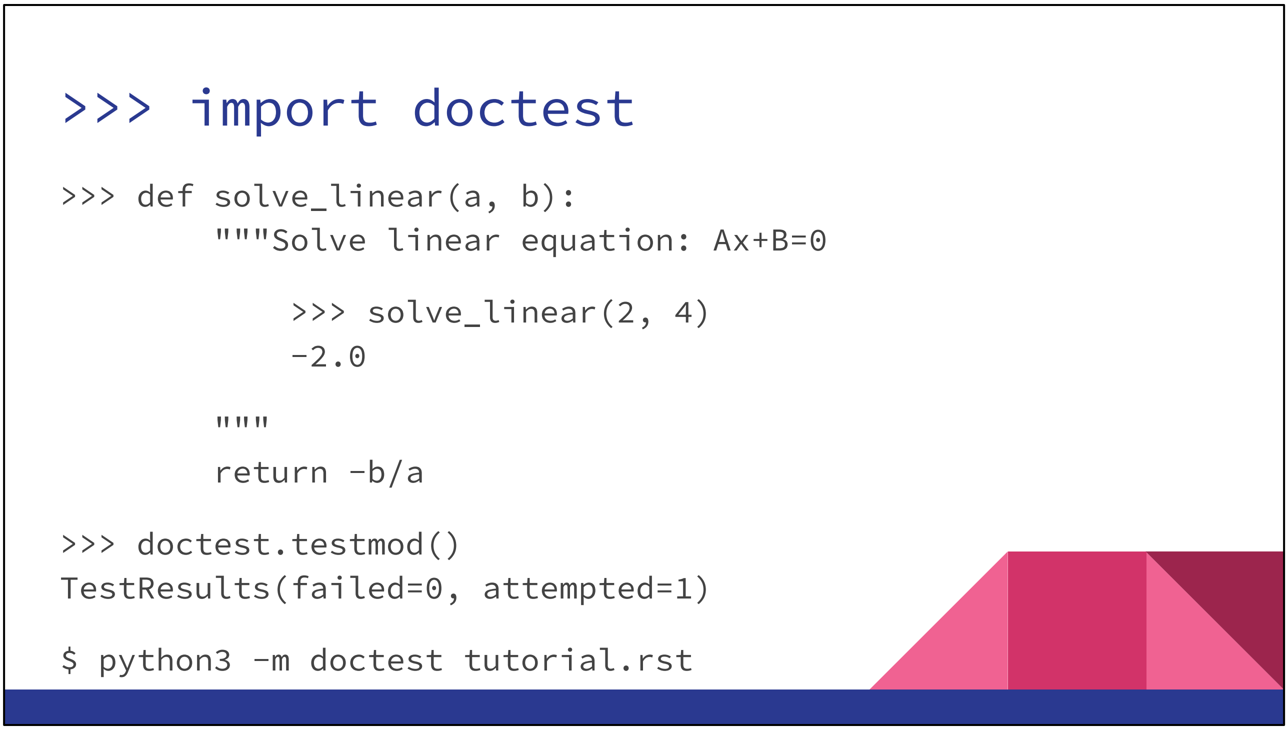
```
>>> import pydoc
```

```
>>> pydoc.help(DocstringDemo.demo)
Help on function demo in module __main__:
demo(self)
    Saved in __doc__ attribute.
```

```
$ python3 -m pydoc sys
```

```
$ python3 -m pydoc -b
```

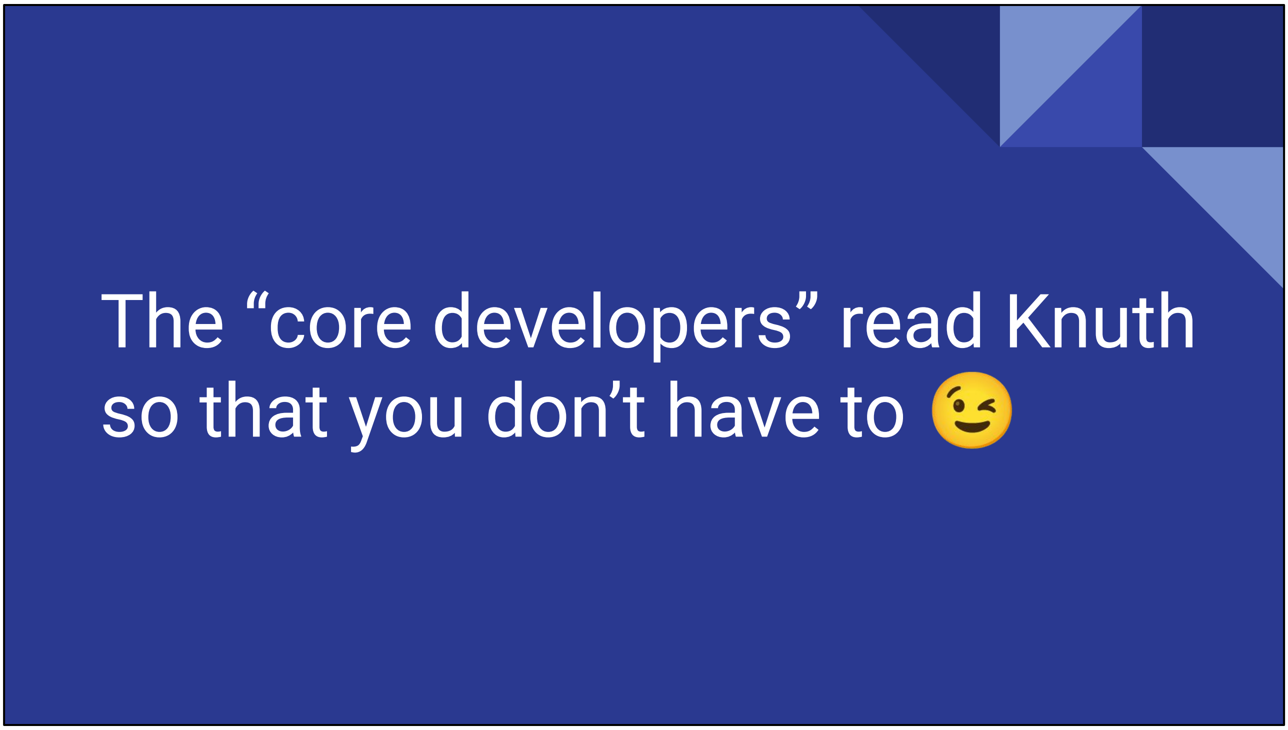
The hard part of pydoc.help() is mostly text formatting.
Also useful for looking at a man-page like view of a module.
Also useful for fancy browser-based documentation.



>>> import doctest

```
>>> def solve_linear(a, b):
        """Solve linear equation: Ax+B=0

            >>> solve_linear(2, 4)
            -2.0

        """
        return -b/a

>>> doctest.testmod()
TestResults(failed=0, attempted=1)

$ python3 -m doctest tutorial.rst
```

It’s a two-fer! Documentation examples (which is all people read anyway) that becomes executable tests.



The “core developers” read Knuth so that you don’t have to 😉