

Literate Programming

in the

Twenty-first Century

A person in a dark, futuristic suit with a glowing visor and a small light on their leg, holding a glowing green device, standing in a dark room with arched windows and a globe on a stand.

Howard Abrams
www.howardism.org
[@howardabrams](https://twitter.com/howardabrams)

Thesis



Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on **explaining to human beings** what we want a computer to do.

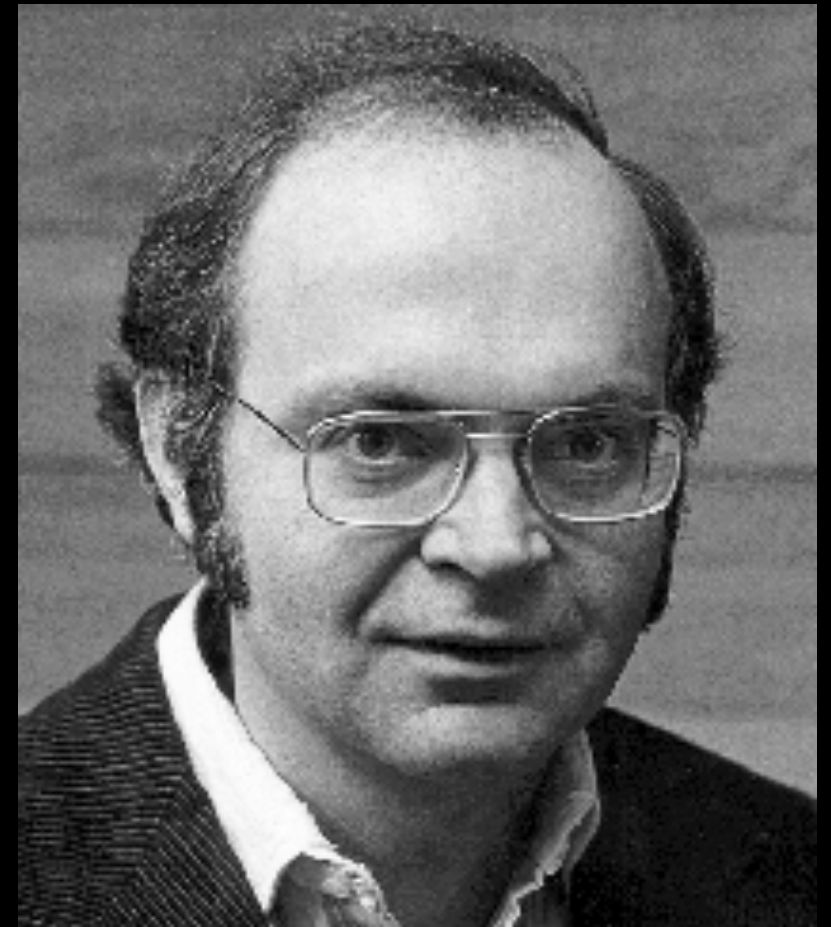
—Donald Knuth



The programmer's task is to state [the] parts and relationships, in whatever order is best for human comprehension not in some rigidly determined order like top-down or bottom-up.

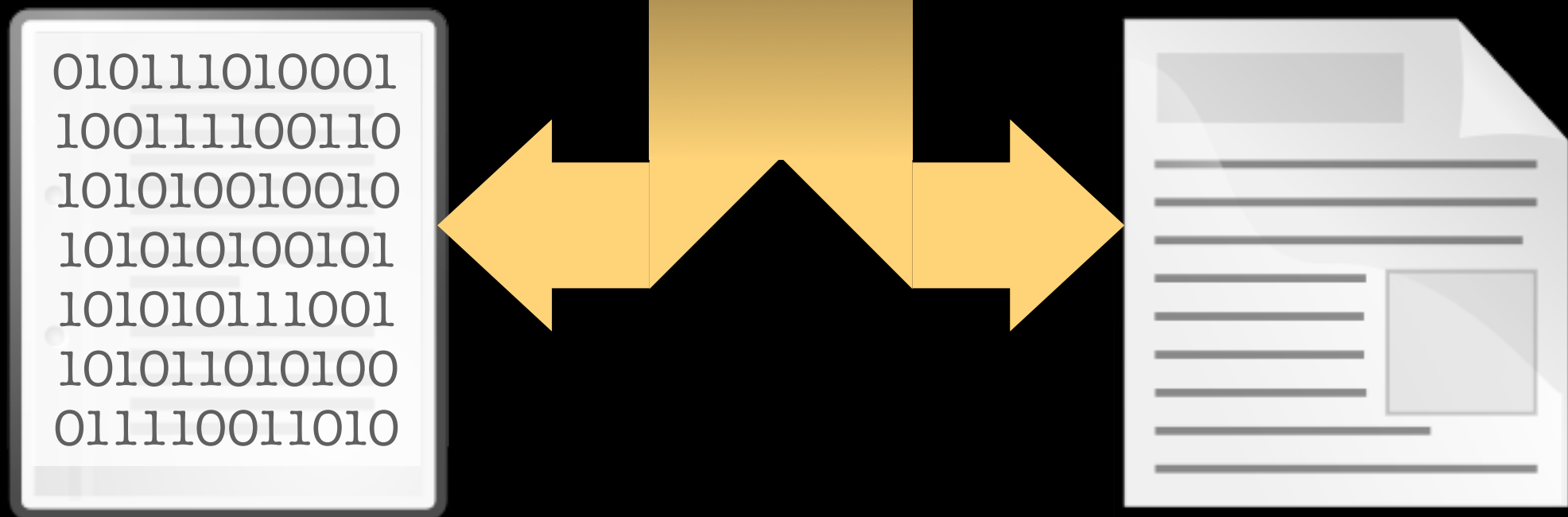
—Donald Knuth

Computer programming is an art...
especially because it produces
objects of beauty. A programmer
who subconsciously views himself
as an artist will enjoy what he does
and will do it better.



—Donald Knuth

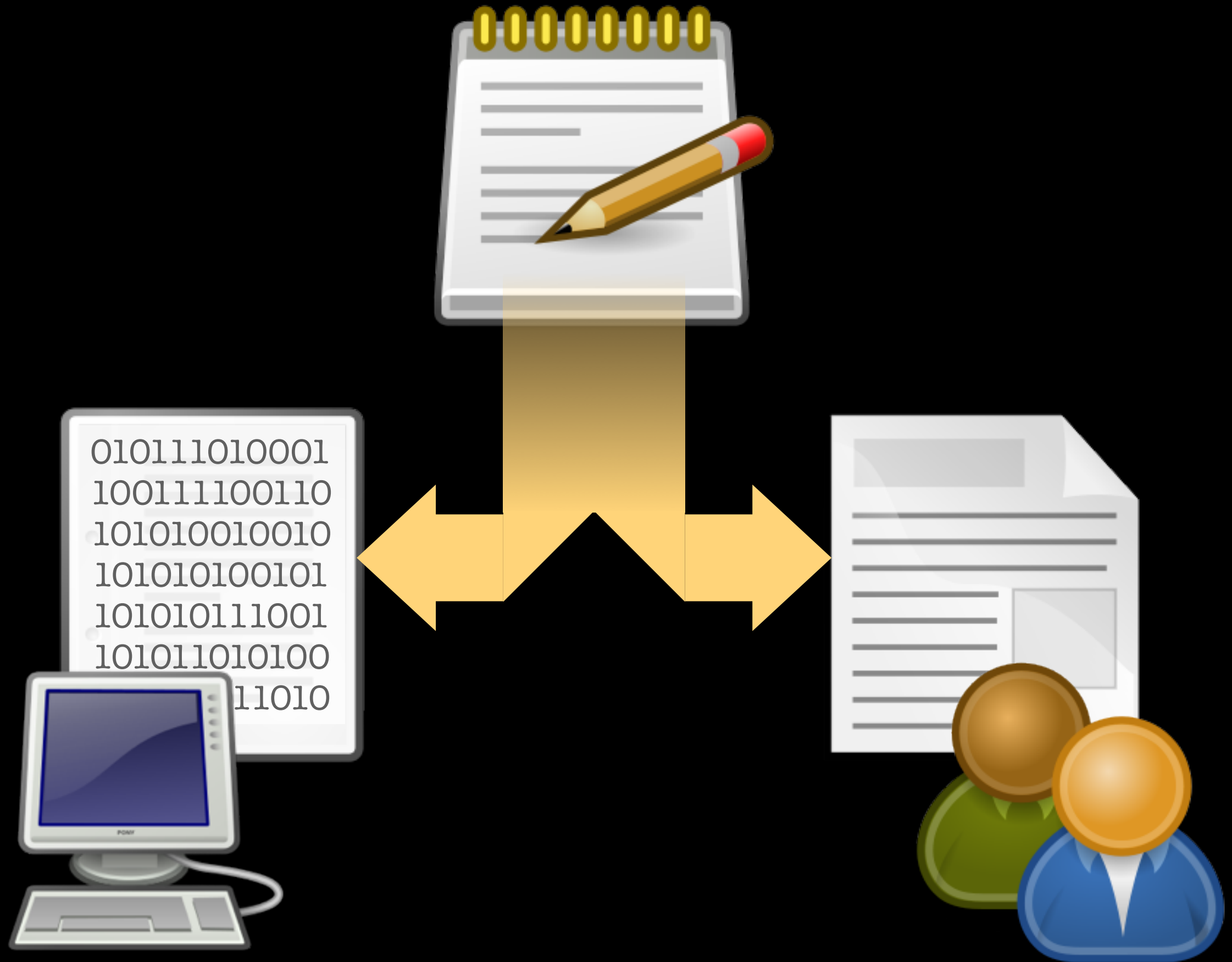




A diagram illustrating a process flow. At the top center is a notepad with a yellow pencil resting on it. A thick yellow arrow points downwards from the notepad, then splits into two arrows pointing left and right. On the left is a rectangular box containing eight lines of binary code. On the right is a document icon with horizontal lines and a small square placeholder. The entire diagram is set against a black background.

```
010111010001
100111100110
101010010010
101010100101
101010111001
101011010100
011110011010
```





Tangling



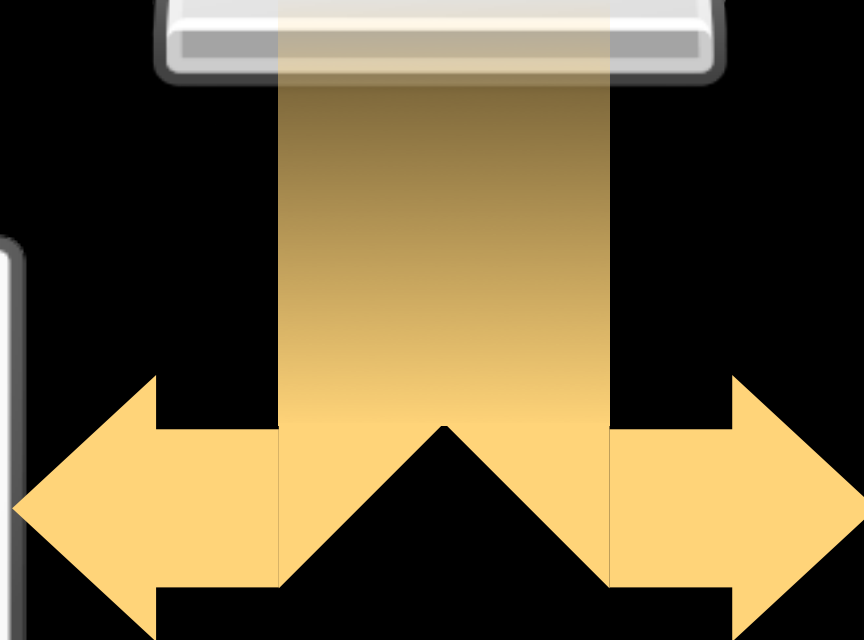


Tangling

```
010111010001
100111100110
101010010010
101010100101
101010111001
101011010100
11010
```




Weaving



We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.



```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

Start of code block marker

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;
```

@

End of code block

Start of code block marker

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
    total_line_count,  
    total_char_count;
```

@

End of code block

Code

Start of code block marker

The diagram illustrates the placement of code block markers. A large curved arrow labeled 'Start of code block marker' points from the bottom text to the beginning of the code block. A smaller curved arrow labeled 'End of code block' points from the bottom text to the '@' symbol. A third curved arrow labeled 'Code' points from the bottom text to the code block itself.

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
    total_line_count,  
    total_char_count;
```

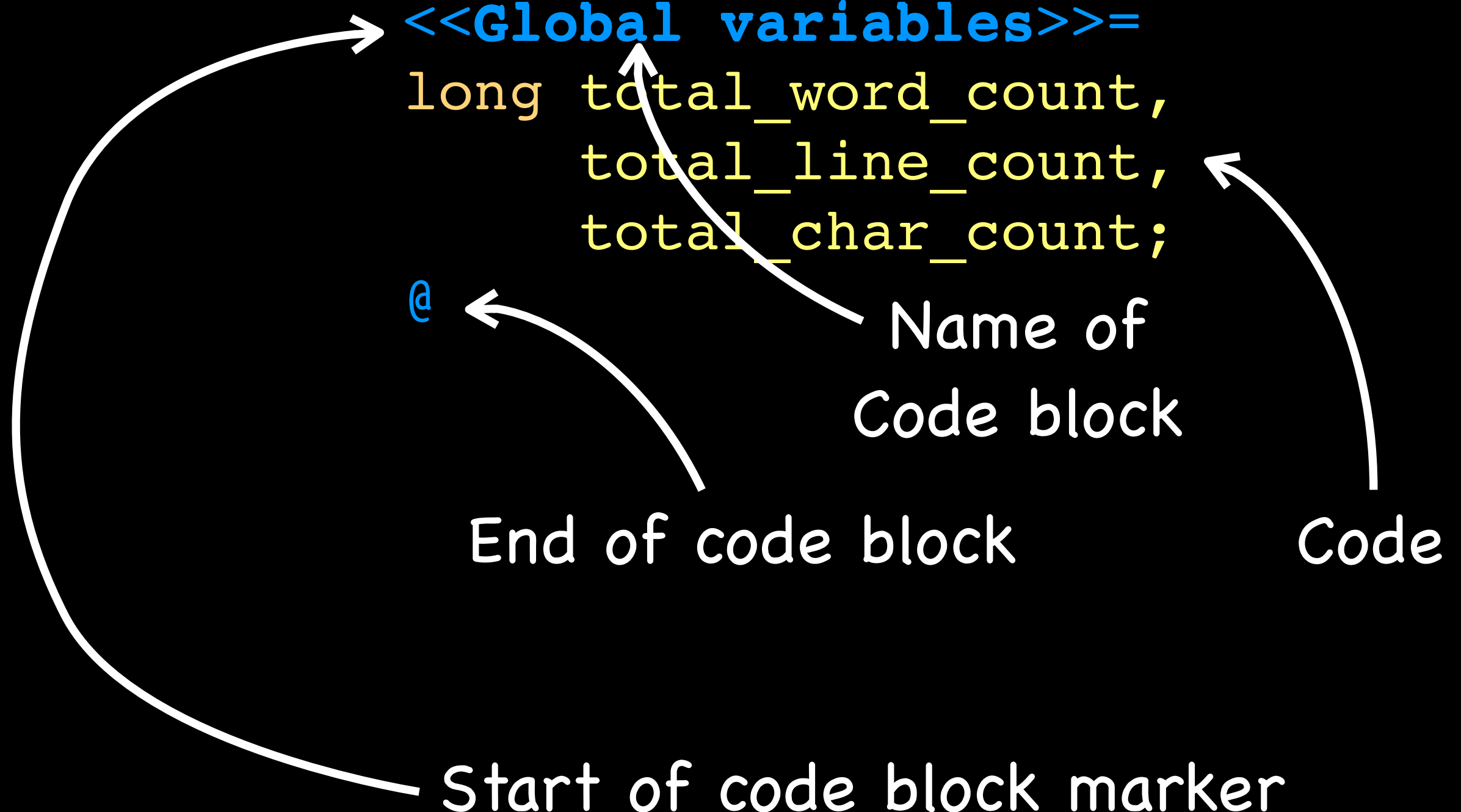
@

Name of
Code block

End of code block

Code

Start of code block marker



We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```


The purpose of wc is to count lines, words, and/or characters in a list of files. The number of lines in a file is ...

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
total_line_count,  
total_char_count;  
@
```

Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:

```
<<*>>=
```

```
<<Header files to include>>
```

```
<<Definitions>>
```

```
<<Global variables>>
```

```
<<Functions>>
```

```
<<The main program>>
```

```
@
```



We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

The purpose of wc is to count lines, words, and/or characters in a list of files. The number of lines in a file is ...

Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:

```
<<*>>=  
    <<Header files to include>>  
    <<Definitions>>  
    <<Global variables>>  
    <<Functions>>  
    <<The main program>>  
@
```

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

The purpose of wc is to count lines, words, and/or characters in a list of files. The number of lines in a file is ...

Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:

```
<<*>>=  
    <<Header files to include>>  
    <<Definitions>>  
    <<Global variables>>  
    <<Functions>>  
    <<The main program>>  
@
```

The purpose of wc is to count lines, words, and/or characters in a list of files. The number of lines in a file is ...


Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

```
<<*>>=  
    <<Header files to include>>  
    <<Definitions>>  
    <<Global variables>>  
    <<Functions>>  
    <<The main program>>
```

```
<<Functions>>=  
    <<Count words in array>>  
    <<Separate words>>  
    <<Is punctuation?>>  
@
```



The purpose of wc is to count lines, words, and/or characters in a list of files. The number of lines in a file is ...

Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:

We must include the standard I/O definitions, since we want to send formatted output to stdout and stderr.

```
<<Global variables>>=  
long total_word_count,  
      total_line_count,  
      total_char_count;  
@
```

```
<<Count words in array>>=  
    // ...  
@
```

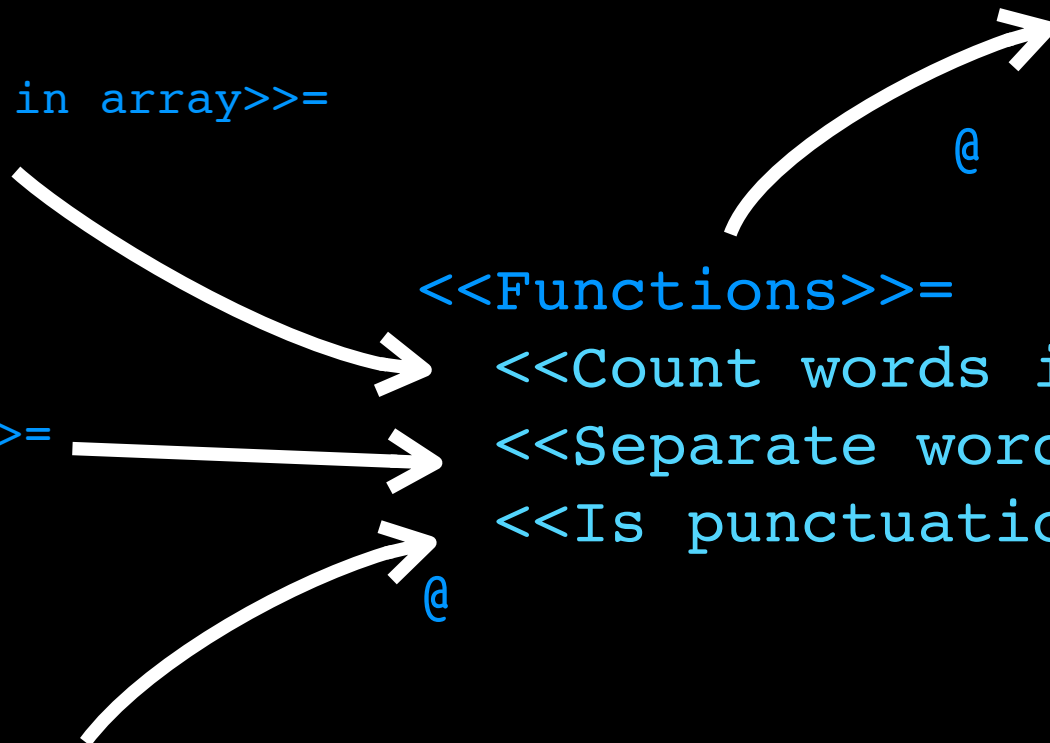
```
<<Separate words>>=  
    // ...  
@
```

```
<<Is punctuation?>>=  
    // ...  
@
```

```
<<Functions>>=  
<<Count words in array>>  
<<Separate words>>  
<<Is punctuation?>>  
@
```

```
<<*>>=
```

```
<<Header files to include>>  
<<Definitions>>  
<<Global variables>>  
<<Functions>>  
<<The main program>>  
@
```



Literate Programming

Donald E. Knuth

Computer Science Department, Stanford University, Stanford, CA 94305, USA

The author and his associates have been experimenting for the past several years with a programming language and documentation system called WEB. This paper presents WEB by example, and discusses why the new system appears to be an improvement over previous ones.

A. INTRODUCTION

The past ten years have witnessed substantial improvements in programming methodology. This advance, carried out under the banner of "structured programming," has led to programs that are more reliable and easier to comprehend; yet the results are not entirely satisfactory. My purpose in the present paper is to propose another motto that may be appropriate for the next decade, as we attempt to make further progress in the state of the art. I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. Hence, my title: "Literate Programming."

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

Advantages of literate programming can be re-

I would ordinarily have assigned to student research assistants; and why? Because it seems to me that at last I'm able to write programs as they should be written. My programs are not only explained better than ever before; they also are better programs, because the new methodology encourages me to do a better job. For these reasons I am compelled to write this paper, in hopes that my experiences will prove to be relevant to others.

I must confess that there may also be a bit of malice in my choice of a title. During the 1970s I was coerced like everybody else into adopting the ideas of structured programming, because I couldn't bear to be found guilty of writing *unstructured* programs. Now I have a chance to get even. By coining the phrase "literate programming," I am imposing a moral commitment on everyone who hears the term; surely nobody wants to admit writing an *illiterate* program.

B. THE WEB SYSTEM

Literate Programming

Donald E. Knuth

Computer Science Department, Stanford University, Stanford, CA 94305

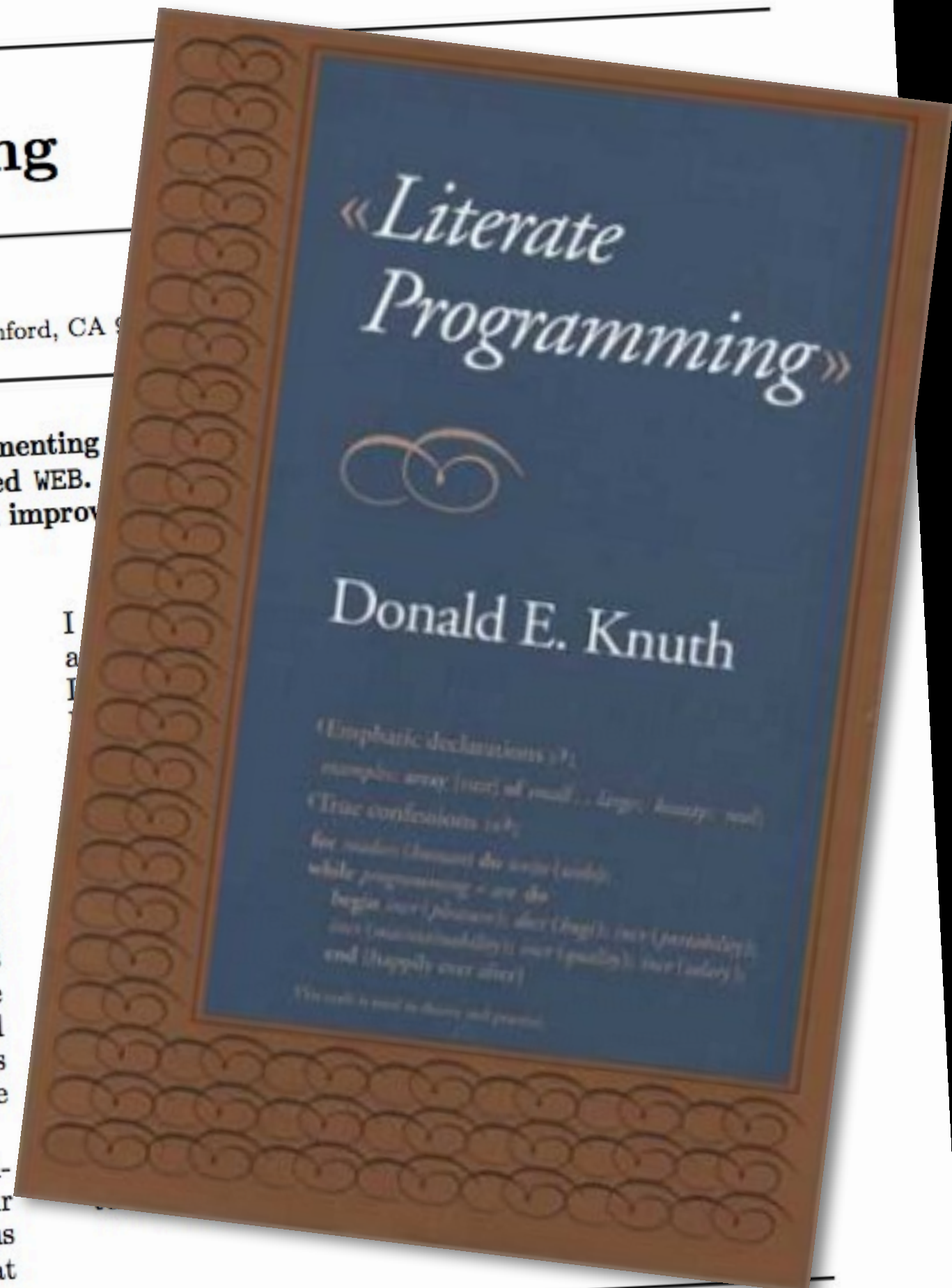
The author and his associates have been experimenting with a programming language and documentation system called WEB. This paper discusses why the new system appears to be an improvement over traditional programming.

A. INTRODUCTION

The past ten years have witnessed substantial improvements in programming methodology. This advance, carried out under the banner of "structured programming," has led to programs that are more reliable and easier to comprehend; yet the results are not entirely satisfactory. My purpose in the present paper is to propose another motto that may be appropriate for the next decade, as we attempt to make further progress in the state of the art. I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. Hence, my title: "Literate Programming."

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

The advantages of literate programming can be re-



B. THE WEB SYSTEM

Antithesis



A wise engineering solution would
produce—or better, exploit—reusable parts.
—Doug McIlory



A wise engineering solution would
produce—or better, exploit—reusable parts.
—Doug McIlory

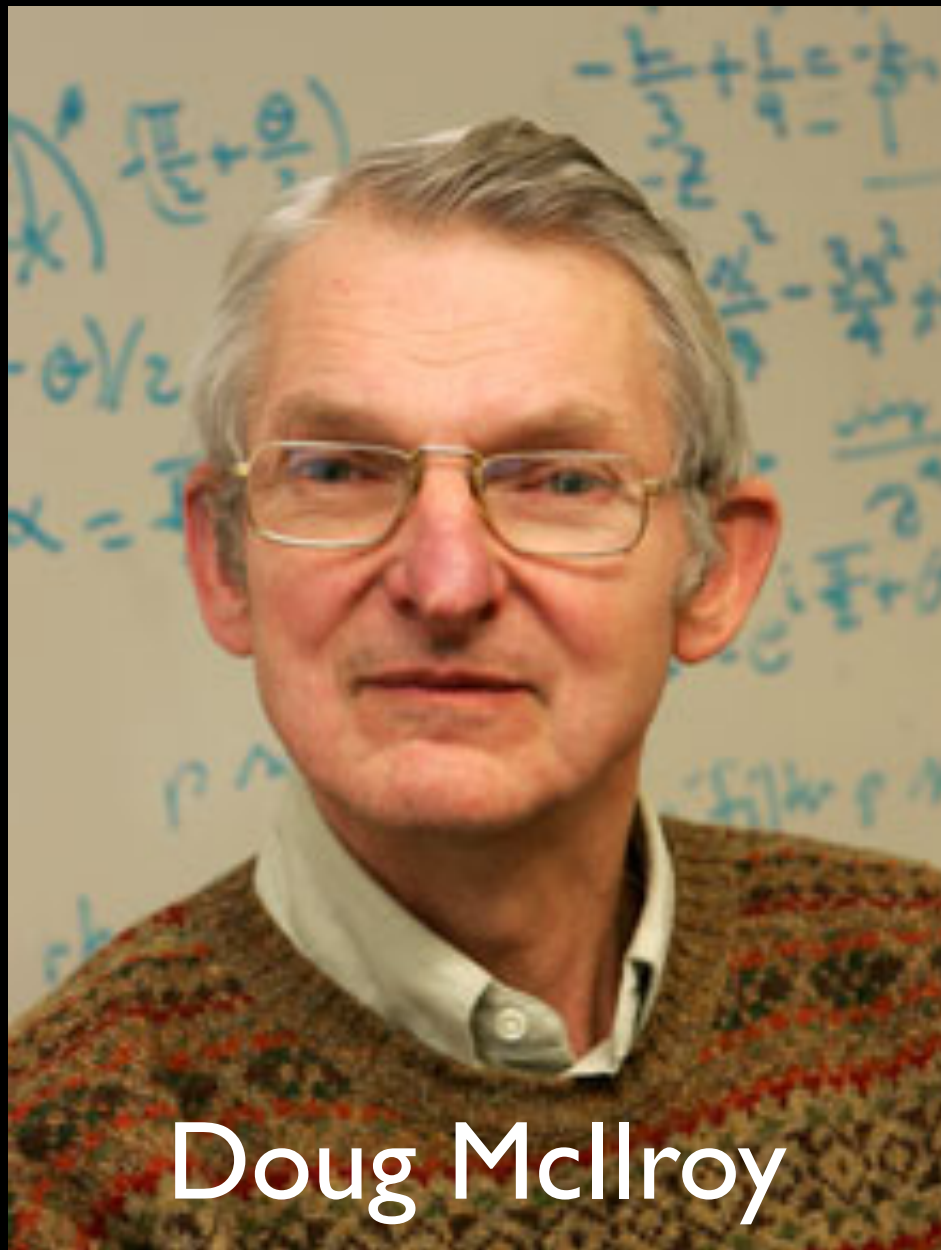
```
tr -cs A-Za-z '\n' |  
tr A-Z a-z |  
sort |  
uniq -c |  
sort -rn |  
sed ${1}q
```



A wise engineering solution would
produce—or better, exploit—reusable parts.
—Doug McIlory

```
tr -cs A-Za-z '\n' |  
tr A-Z a-z |  
sort |  
uniq -c |  
sort -rn |  
sed ${1}q
```

*It's funny, it's
reusable, and it wildly
misses Knuth's point.
—Po Petz*



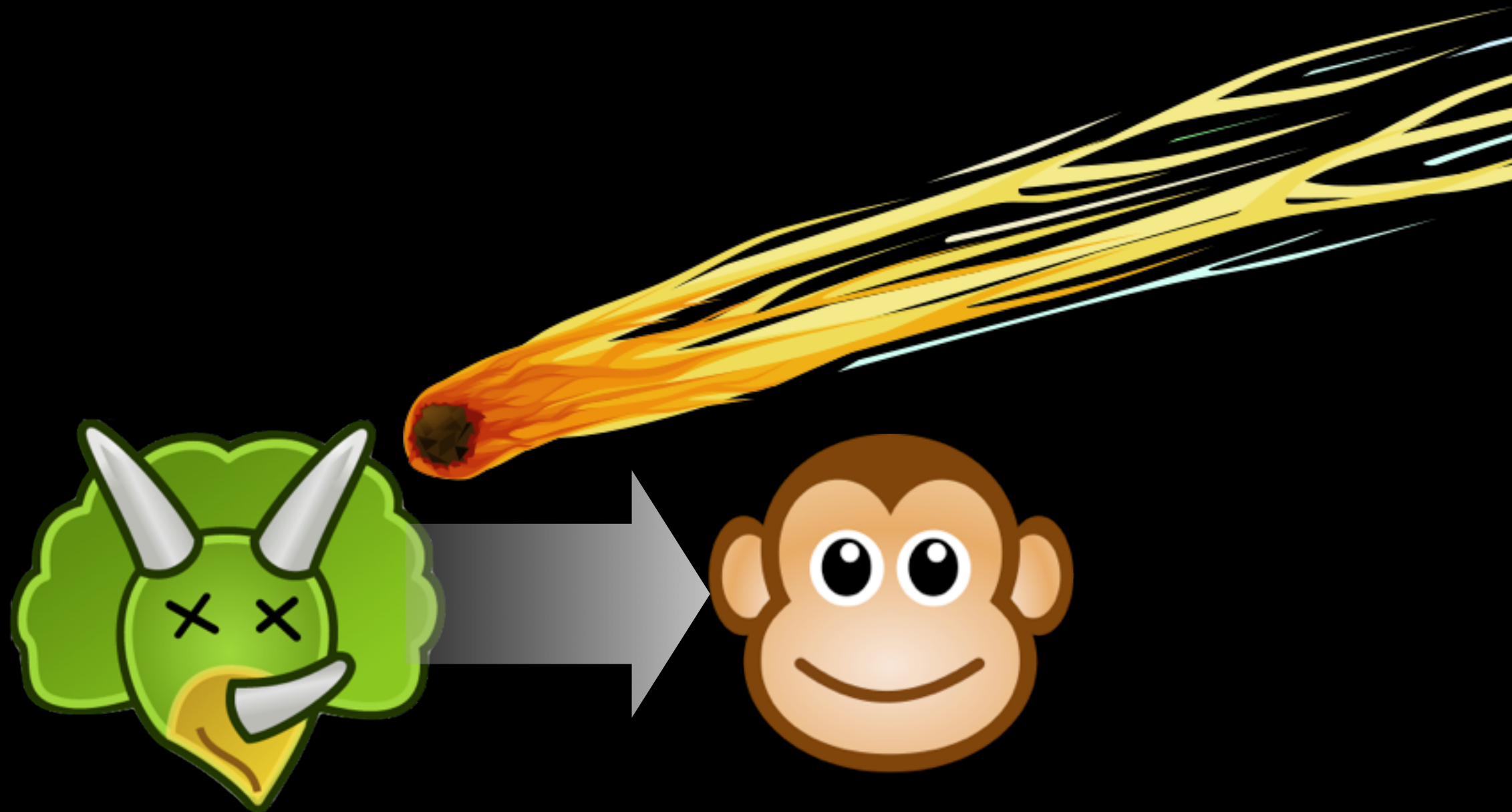
Code re-use



Better thinking through better tools







Influences

- Javadoc System
- Docco
- iPython Notebook
- Haskell
- Embraced by Cryptic Languages

<http://aanandprasad.com/articles/negronis/>

The Abstracted Negroni

This post is written in literate javascript. You can [download it here](#) and run it at the command line thus: `cat negronis.litjs | egrep '^ {4}' | node`

I was out last Friday at a bar where they had a “Negroni Tic-Tac-Toe” offer—you could custom-build your drink from a selection of 3 gins, 3 vermouths and 3 amari, and if you got “3 in a row” you’d get £5 off your bill. It’s a laughably stingy deal, but it got me thinking. About programming, I mean.

```
function Negroni(gin, vermouth, amaro) {  
  this.gin      = gin;  
  this.vermouth = vermouth;  
  this.amaro    = amaro;  
  
  // Build over ice, stir well  
}
```

<http://aanandprasad.com/articles/negronis/>

The Abstracted Negroni

This post is written in `literate javascript`. You can download it [here](#) and run it at the command line thus: `cat negronis.litjs | egrep '^ {4}' | node`

I was out last Friday at a bar where they had a “Negroni Tic-Tac-Toe” offer—you could custom-build your drink from a selection of 3 gins, 3 vermouths and 3 amari, and if you got “3 in a row” you’d get £5 off your bill. It’s a laughably stingy deal, but it got me thinking. About programming, I mean.

```
function Negroni(gin, vermouth, amaro) {  
  this.gin      = gin;  
  this.vermouth = vermouth;  
  this.amaro    = amaro;  
  
  // Build over ice, stir well  
}
```

File Edit View Insert Cell Kernel Help



Parameters

A *parameter* (we sometimes call them *arguments*) are things we can pass *into* a function. For instance:

```
In [5]: def hello(name):  
        print "Hello", name  
  
        hello("Charlie")  
  
        Hello Charlie
```

You can pass in more than one thing into a **function** if you separate them with commas:

```
In [7]: def larger(a, b):  
        if a < b:  
            print a, "is less than", b  
        elif a > b:  
            print a, "is greater than", b  
        else:  
            print a, "is the same as", b  
  
        larger(3, 5)  
  
        3 is less than 5
```


File Edit View Insert Cell Kernel Help



Parameters

A *parameter* (we sometimes call them *arguments*) are things we can pass *into* a function. For instance:

```
In [5]: def hello(name):  
        print "Hello", name  
  
        hello("Charlie")  
  
        Hello Charlie
```

You can pass in more than one thing into a **function** if you separate them with commas:

```
In [7]: def larger(a, b):  
        if a < b:  
            print a, "is less than", b  
        elif a > b:  
            print a, "is greater than", b  
        else:  
            print a, "is the same as", b  
  
        larger(3, 5)  
  
        3 is less than 5
```

A page is a series
of “cells”

File Edit View Insert Cell Kernel Help

A toolbar containing icons for file operations (save, copy, paste), navigation (up, down, first, last), and execution (run, interrupt). To the right of the icons is a dropdown menu currently showing 'Markdown'.

Parameters

A *parameter* (we sometimes call them *arguments*) are things we can pass *into* a function. For instance:

```
In [5]: def hello(name):  
        print "Hello", name  
  
        hello("Charlie")  
        Hello Charlie
```

Executed code is
displayed below

Two blue arrows originate from the text 'Executed code is displayed below'. One arrow points to the function definition code in the previous block, and the other points to the output 'Hello Charlie'.

You can pass in more than one thing into a **function** if you separate them with commas:

```
In [7]: def larger(a, b):  
        if a < b:  
            print a, "is less than", b  
        elif a > b:  
            print a, "is greater than", b  
        else:  
            print a, "is the same as", b  
  
        larger(3, 5)  
  
        3 is less than 5
```

File Edit View Insert Cell Kernel Help

A set of small, light-gray icons for notebook navigation and editing, including symbols for saving, undo, redo, and running code.

Parameters

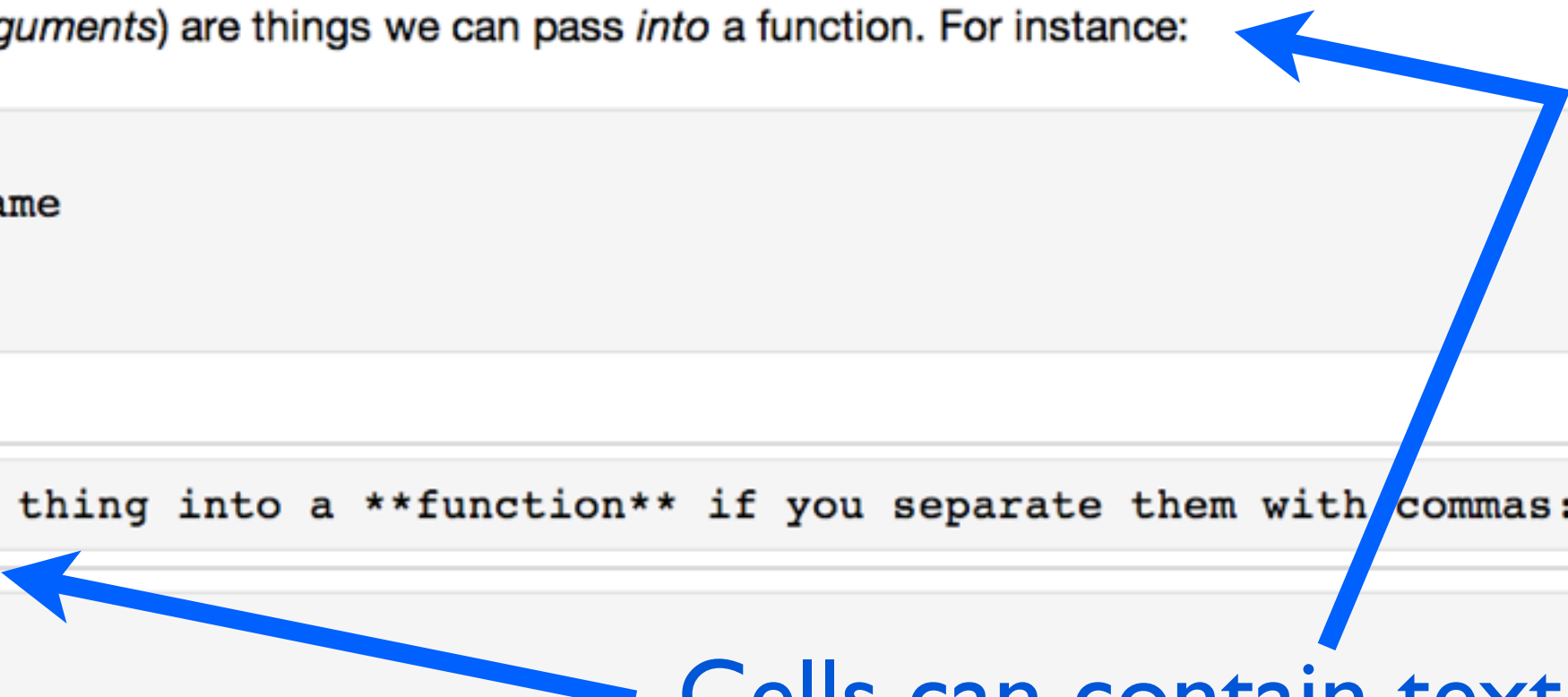
A *parameter* (we sometimes call them *arguments*) are things we can pass *into* a function. For instance:

```
In [5]: def hello(name):  
        print "Hello", name  
  
        hello("Charlie")  
  
        Hello Charlie
```

You can pass in more than one thing into a **function** if you separate them with commas:

```
In [7]: def larger(a, b):  
        if a < b:  
            print a, "is less than", b  
        elif a > b:  
            print a, "is greater than", b  
        else:  
            print a, "is the same as", b  
  
        larger(3, 5)  
  
        3 is less than 5
```

Cells can contain text
in Markdown format,
which is automatically
rendered.

Two blue arrows originate from the text block on the right. One arrow points to the first text cell (the paragraph about parameters), and the other points to the second text cell (the paragraph about passing multiple arguments).

Synthesis

What is Needed?

- Good text processing *and* programming
- Identify and separate source code snippets
- Code block evaluation support
- Link and reference code block snippets
- Use evaluated code output
- Render both code and documentation



In the third millenium, does it still make sense to work with text files? Text files are the only truly portable format for files. The data will never get lost.

—Carsten Dominik



Tangling

```
010111010001
100111100110
101010010010
101010100101
101010111001
101011010100
11010
```



Weaving





Tangling

```
010111010001
100111100110
101010010010
101010100101
101010111001
101011010100
11010
```



Weaving



Connect to Interpreters

[Redacted text block]

- [Redacted list item]
- [Redacted list item]
- [Redacted list item]
- [Redacted list item]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

Prose

Code

Prose

Code

Prose

Code

Prose

Lists, tables and
textual data fed
in as variables



Lists, tables and
textual data fed
in as variables



[Redacted text block]

- [Redacted]
- [Redacted]
- [Redacted]
- [Redacted]

[Redacted text block]

- [Redacted]
- [Redacted]
- [Redacted]
- [Redacted]
- [Redacted]
- [Redacted]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]



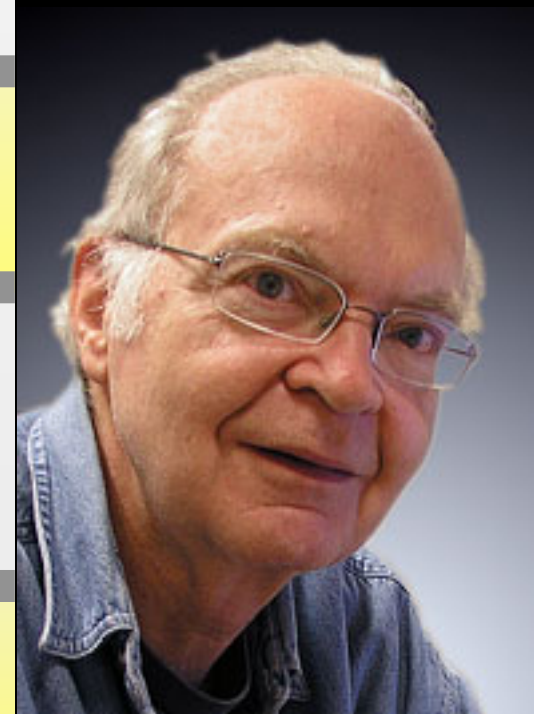
Results of
running code
inserted as data



Results of *that*
code given as
variables to other
code blocks

A complex piece of software is best regarded as *a web of ideas* that has been delicately pieced together from simple materials.

—Knuth





Knuth originally
interconnected code.

Now we can
interconnect both
code and data in a
literate way.

The Toolbox

Babel

org-mode

Emacs

Language Modes

Graphviz/PlantUML

REPL Connectors

Demonstration

Possible Uses

- Learning a new language or technology
- Better REPL for non-interactive languages
- Problems require multiple languages
- Embedded UML or other diagrams
- Combining code with its tests
- Easier to brain-storm over complex analysis
- Describe complex code:
 - Regular Expressions
 - Odd inheritance trees
 - SQL and ORM

Questions?

Links to this presentation and other
bookmarks available at either this
URL or scan this QR code:

<http://is.gd/XPGMR6>

