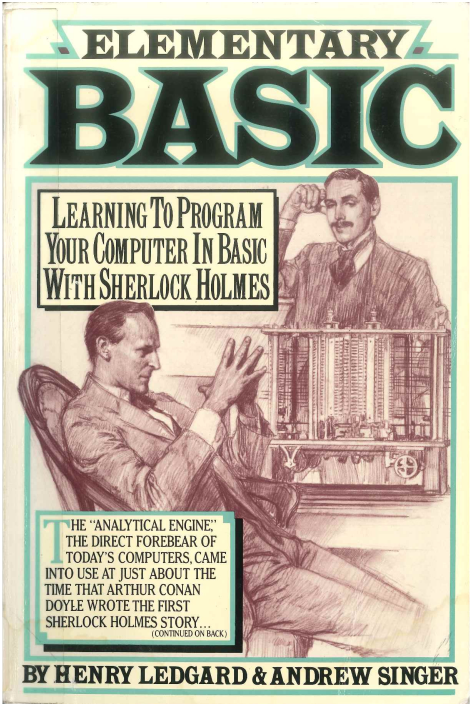
What can a 1980s BASIC programming textbook teach us today?

Martin Lester

Department of Computer Science,
University of Oxford

History and Philosophy of Programming,
2018–03–23


ELEMENTARY
BASIC
LEARNING TO PROGRAM YOUR COMPUTER IN BASIC WITH SHERLOCK HOLMES
THE "ANALYTICAL ENGINE," THE DIRECT FOREBEAR OF TODAY'S COMPUTERS, CAME INTO USE AT JUST ABOUT THE TIME THAT ARTHUR CONAN DOYLE WROTE THE FIRST SHERLOCK HOLMES STORY...
(CONTINUED ON BACK)
BY HENRY LEDGARD & ANDREW SINGER

Motivation

- *Elementary Basic: Learning to Program your Computer in Basic with Sherlock Holmes* is an introductory book on programming from 1982 with an interesting central conceit . . .
- I have seen several introductory programming books and articles from the 1980s, but none like this.
- Looking beyond the presentation, the technical content and its structure is atypical too.
- What can we learn from an unusual source like this?
- How does it compare with modern books or with other books of its time?

Central Conceit

- Sherlock Holmes used Charles Babbage's Analytical Engine to assist in solving various mysteries.
- The authors have supposedly discovered unpublished manuscripts detailing Watson's discussions with Holmes on these ventures into programming.
- The manuscripts have been translated into BASIC for the benefit of the modern reader.

Outline

Historical Context

- By the early 1980s, commercially produced home computers were available to the public at reasonable prices:
  - 1977 Trinity — Apple II, Commodore PET, TRS-80
  - 1982 — Commodore 64, ZX Spectrum, BBC Micro
- These computers came with BASIC built in.
- The BASIC interpreter typically doubled as the operating system.
- Books and articles in magazines on how to program were widespread.
- Authors of Elementary Basic questioned *whether the world really needed yet another text on programming*.

Structure of the Book

- Structure of the book: sequence of increasingly complex programs.
    - Not organised around introduction of standard language features!
- Structure of a typical chapter:
    - After some setup, Holmes discusses with Watson an aspect of a mystery that could be solved easily using a computer.
    - Holmes presents pseudo-code for solving the problem, followed by a BASIC implementation.
    - Watson asks questions about new ideas or confusing parts of the program, which Holmes answers.
    - The chapter concludes with an "out of character" summary of any new language features introduced.

About the Authors

Who wrote this book?

- Henry Ledgard
  - PhD in CS from MIT; postdoc at Oxford
  - Part of the ADA design team
- Andrew Singer
  - PhD in CS from Massachusetts
  - Was an editor of short-lived ROM magazine
- Non-credited authors played substantial roles in book's development . . .
  - . . . in particular in writing the pastiches of Sherlock Holmes stories.

Lesson: If you want to write an interesting book for a general audience, you might need someone skilled primarily as a writer.

Motivation of the Authors

The authors claimed that:

- *programming is difficult...[but] the basis of programming stems from a few elementary ideas*;
- the dialogue format of the book is an *easy to follow and enjoyable* exposition of those ideas;
- and *the best method to teach programming is through problems*.

Concept and Format

The key idea behind the book is certainly novel . . .

- . . . and it makes people want to read it.

The dialogue format is also unusual.

- The idea of using a dialogue to expose ideas is very old.
- It allows possible points of contention or causes of confusion to be explored in a natural way.
- Why is it not used more often? (Not just in books.)

Problems to Motivate Programming

Problems that can be solved using a computer are presented and solved.

- Can be more engaging than typical “how to add a list of numbers” example programs.
- Would it have been more engaging to its audience than writing a text adventure or Space Invaders clone?
- Not necessarily the case that new programs introduce new language features; it might just be increased complexity.
- Distinct approach from most introductory programming texts then and now.

Note this is not *problem-based learning*:

- Little attempt to engage reader in exploratory thinking and problem solving.
- No suggested exercises for the reader to try.

Problems to Motivate Programming

Here are some of the programs from the book:

- Solving a murder from clues.
- Calculating number of tide cycles between dates and times.
- Turning Julian days into date and month.
- Calculating molecular weight of a compound from its formula.
- Formatting and displaying a coroner's report.
- Searching a flat file database for a matching criminal record.

Although the setting is interesting, many of the programs end up being rather pedestrian.

Focus on Program Development

Types:

- Types as enforced by the language.
- Types as intended by the programmer.

Top-down design:

- In vogue at the time.
- Emphasised throughout the book.
- Rare to discuss design methods much at all in other introductory books of the time.

Code hygiene:

- Use descriptive variable names.
- Add comments.
- Use indentation meaningfully.

How BASIC Hinders the Book

BASIC was the most popular language of its day with good reason:

- Relatively easy to implement an interpreter in a computer with little memory and processing power.
- English-like keywords make it less frightening to beginners.
- Lack of enforced structure means you can start programming immediately.

Today, it is popular to criticise BASIC. That is not my intention. Nonetheless:

- Some aspects of BASIC make it difficult to write complex programs.
- This shows the choice of language is significant in introducing programming.
- There was also a version of this book in Pascal.
    - Was it any good?
    - Did anyone read it?

Focus on Program Development (revisited)

Types:

- Types as enforced by the language. Very few language types.
- Types as intended by the programmer.

Top-down design:

- In vogue at the time.
- Emphasised throughout the book.
- Rare to discuss design methods much at all in other introductory books of the time.
- Difficult to add lines in middle of program — difficult to refine.

Code hygiene:

- Use descriptive variable names. Some BASICs only support short names.
- Add comments. Uses up valuable memory.
- Use indentation meaningfully. Discarded by many implementations.

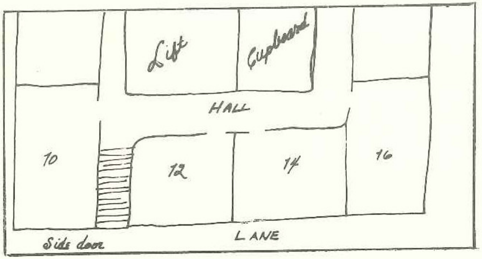
How BASIC Hinders the Book (continued)

One of the most popular criticisms of BASIC is its emphasis on GOTO. But:

- Sometimes useful if you don't have the high-level control structure you want.
- You can still do structured programming — as this book shows.
- Many implementations supported for/while loops.
- Being able to jump is a necessary concession to difficulty of inserting lines.

In my opinion, limited or no support for compound datatypes or dynamic allocation of data structures is a far bigger disadvantage.

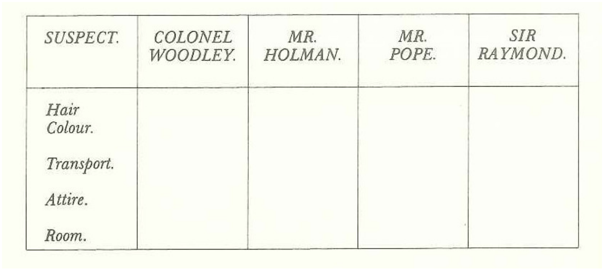
Solving a Murder: The Problem



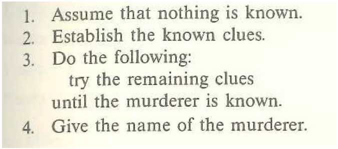
The clues were as follows :

1. Sir Raymond Jasper occupied Room 10.
2. The man occupying Room 14 had black hair.
3. Either Colonel Woodley or Sir Raymond wore a pince-nez.
4. Mr. Pope always carried a gold pocket watch.
5. One of the suspects was seen driving a four-wheel carriage.
6. The man with the pince-nez had brown hair.
7. Mr. Holman wore a ruby signet ring.
8. The man in Room 16 had tattered cuffs.
9. Mr. Holman occupied Room 12.
10. The man with tattered cuffs had red hair.
11. The man in Room 12 had grey hair.
12. The man with a gold pocket watch occupied Room 14.
13. Colonel Woodley occupied a corner room.
14. The murderer had brown hair.

Solving a Murder: Top-Down Design

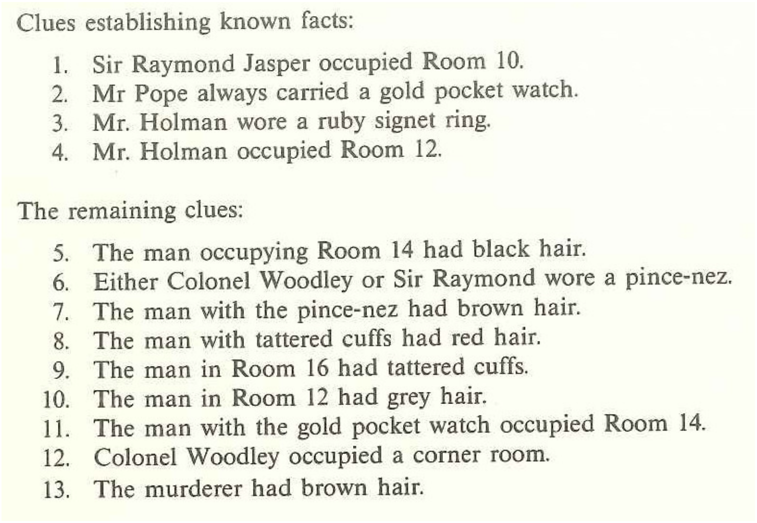


SUSPECT.	*COLONEL WOODLEY.*	*MR. HOLMAN.*	*MR. POPE.*	*SIR RAYMOND.*
Hair Colour.				
Transport.				
Attire.				
Room.				



1. Assume that nothing is known.
2. Establish the known clues.
3. Do the following:
   try the remaining clues
   until the murderer is known.
4. Give the name of the murderer.

Solving a Murder: Classifying the Clues



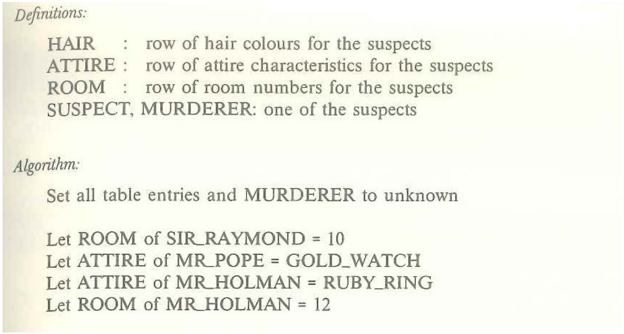
Clues establishing known facts:

1. Sir Raymond Jasper occupied Room 10.
2. Mr Pope always carried a gold pocket watch.
3. Mr. Holman wore a ruby signet ring.
4. Mr. Holman occupied Room 12.

The remaining clues:

5. The man occupying Room 14 had black hair.
6. Either Colonel Woodley or Sir Raymond wore a pince-nez.
7. The man with the pince-nez had brown hair.
8. The man with tattered cuffs had red hair.
9. The man in Room 16 had tattered cuffs.
10. The man in Room 12 had grey hair.
11. The man with the gold pocket watch occupied Room 14.
12. Colonel Woodley occupied a corner room.
13. The murderer had brown hair.

Solving a Murder: “Algorithm”



Definitions:

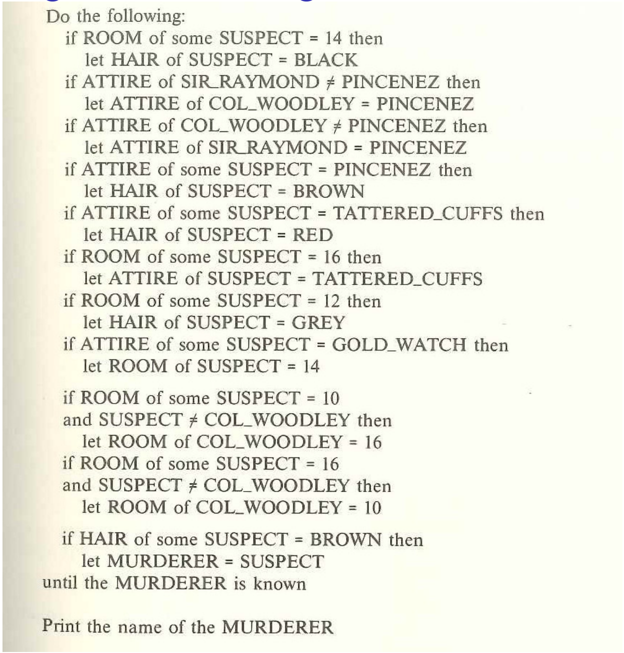
HAIR	:	row of hair colours for the suspects
ATTIRE	:	row of attire characteristics for the suspects
ROOM	:	row of room numbers for the suspects

SUSPECT, MURDERER: one of the suspects

Algorithm:

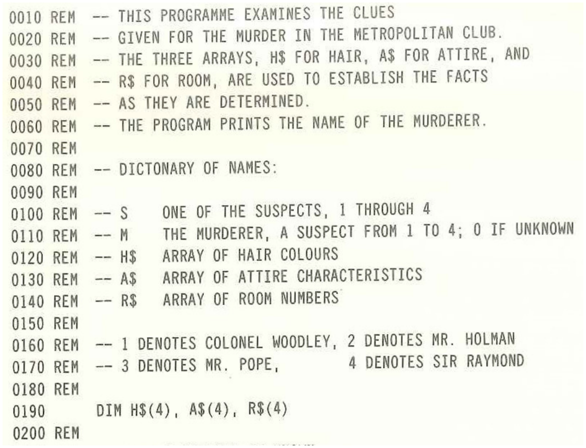
```
Set all table entries and MURDERER to unknown

Let ROOM of SIR_RAYMOND = 10
Let ATTIRE of MR_POPE = GOLD_WATCH
Let ATTIRE of MR_HOLMAN = RUBY_RING
Let ROOM of MR_HOLMAN = 12
```

Solving a Murder: “Algorithm”



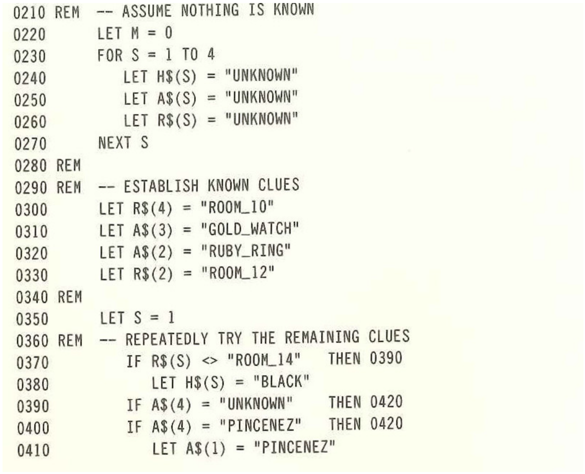
```
Do the following:
  if ROOM of some SUSPECT = 14 then
    let HAIR of SUSPECT = BLACK
  if ATTIRE of SIR_RAYMOND ≠ PINCENEZ then
    let ATTIRE of COL_WOODLEY = PINCENEZ
  if ATTIRE of COL_WOODLEY ≠ PINCENEZ then
    let ATTIRE of SIR_RAYMOND = PINCENEZ
  if ATTIRE of some SUSPECT = PINCENEZ then
    let HAIR of SUSPECT = BROWN
  if ATTIRE of some SUSPECT = TATTERED_CUFFS then
    let HAIR of SUSPECT = RED
  if ROOM of some SUSPECT = 16 then
    let ATTIRE of SUSPECT = TATTERED_CUFFS
  if ROOM of some SUSPECT = 12 then
    let HAIR of SUSPECT = GREY
  if ATTIRE of some SUSPECT = GOLD_WATCH then
    let ROOM of SUSPECT = 14

  if ROOM of some SUSPECT = 10
  and SUSPECT ≠ COL_WOODLEY then
    let ROOM of COL_WOODLEY = 16
  if ROOM of some SUSPECT = 16
  and SUSPECT ≠ COL_WOODLEY then
    let ROOM of COL_WOODLEY = 10

  if HAIR of some SUSPECT = BROWN then
    let MURDERER = SUSPECT
until the MURDERER is known

Print the name of the MURDERER
```

Solving a Murder: Code



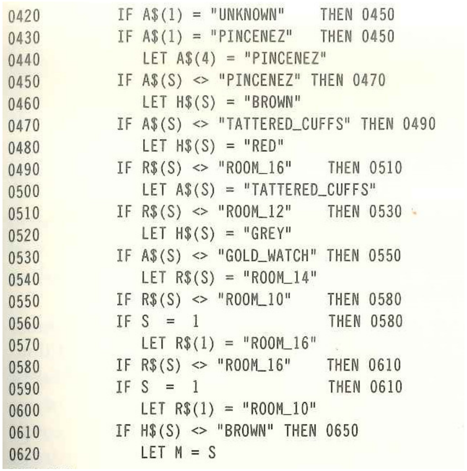
```
0010 REM  -- THIS PROGRAMME EXAMINES THE CLUES
0020 REM  -- GIVEN FOR THE MURDER IN THE METROPOLITAN CLUB.
0030 REM  -- THE THREE ARRAYS, H$ FOR HAIR, A$ FOR ATTIRE, AND
0040 REM  -- R$ FOR ROOM, ARE USED TO ESTABLISH THE FACTS
0050 REM  -- AS THEY ARE DETERMINED.
0060 REM  -- THE PROGRAM PRINTS THE NAME OF THE MURDERER.
0070 REM
0080 REM  -- DICTONARY OF NAMES:
0090 REM
0100 REM  -- S    ONE OF THE SUSPECTS, 1 THROUGH 4
0110 REM  -- M    THE MURDERER, A SUSPECT FROM 1 TO 4; 0 IF UNKNOWN
0120 REM  -- H$   ARRAY OF HAIR COLOURS
0130 REM  -- A$   ARRAY OF ATTIRE CHARACTERISTICS
0140 REM  -- R$   ARRAY OF ROOM NUMBERS
0150 REM
0160 REM  -- 1 DENOTES COLONEL WOODLEY, 2 DENOTES MR. HOLMAN
0170 REM  -- 3 DENOTES MR. POPE,        4 DENOTES SIR RAYMOND
0180 REM
0190      DIM H$(4), A$(4), R$(4)
0200 REM
```

Solving a Murder: Code



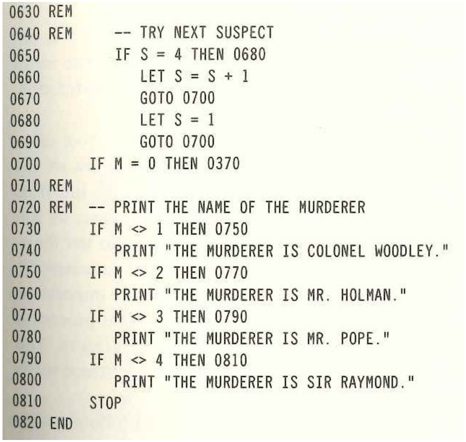
```
0210 REM  -- ASSUME NOTHING IS KNOWN
0220      LET M = 0
0230      FOR S = 1 TO 4
0240         LET H$(S) = "UNKNOWN"
0250         LET A$(S) = "UNKNOWN"
0260         LET R$(S) = "UNKNOWN"
0270      NEXT S
0280 REM
0290 REM  -- ESTABLISH KNOWN CLUES
0300      LET R$(4) = "ROOM_10"
0310      LET A$(3) = "GOLD_WATCH"
0320      LET A$(2) = "RUBY_RING"
0330      LET R$(2) = "ROOM_12"
0340 REM
0350      LET S = 1
0360 REM  -- REPEATEDLY TRY THE REMAINING CLUES
0370         IF R$(S) <> "ROOM_14"   THEN 0390
0380            LET H$(S) = "BLACK"
0390         IF A$(4) = "UNKNOWN"    THEN 0420
0400         IF A$(4) = "PINCENEZ"   THEN 0420
0410            LET A$(1) = "PINCENEZ"
```

Solving a Murder: Code



```
0420      IF A$(1) = "UNKNOWN"    THEN 0450
0430      IF A$(1) = "PINCENEZ"   THEN 0450
0440         LET A$(4) = "PINCENEZ"
0450      IF A$(S) <> "PINCENEZ" THEN 0470
0460         LET H$(S) = "BROWN"
0470      IF A$(S) <> "TATTERED_CUFFS" THEN 0490
0480         LET H$(S) = "RED"
0490      IF R$(S) <> "ROOM_16"    THEN 0510
0500         LET A$(S) = "TATTERED_CUFFS"
0510      IF R$(S) <> "ROOM_12"    THEN 0530
0520         LET H$(S) = "GREY"
0530      IF A$(S) <> "GOLD_WATCH" THEN 0550
0540         LET R$(S) = "ROOM_14"
0550      IF R$(S) <> "ROOM_10"    THEN 0580
0560      IF S  =  1               THEN 0580
0570         LET R$(1) = "ROOM_16"
0580      IF R$(S) <> "ROOM_16"    THEN 0610
0590      IF S  =  1               THEN 0610
0600         LET R$(1) = "ROOM_10"
0610      IF H$(S) <> "BROWN" THEN 0650
0620         LET M = S
```

Solving a Murder: Code



```
0630 REM
0640 REM    -- TRY NEXT SUSPECT
0650        IF S = 4 THEN 0680
0660           LET S = S + 1
0670           GOTO 0700
0680           LET S = 1
0690           GOTO 0700
0700     IF M = 0 THEN 0370
0710 REM
0720 REM -- PRINT THE NAME OF THE MURDERER
0730     IF M <> 1 THEN 0750
0740        PRINT "THE MURDERER IS COLONEL WOODLEY."
0750     IF M <> 2 THEN 0770
0760        PRINT "THE MURDERER IS MR. HOLMAN."
0770     IF M <> 3 THEN 0790
0780        PRINT "THE MURDERER IS MR. POPE."
0790     IF M <> 4 THEN 0810
0800        PRINT "THE MURDERER IS SIR RAYMOND."
0810     STOP
0820 END
```

Conclusions

How is programming to be taught?

- Programming books should be written in collaboration with writers.
- Dialogues are an appealing method of development and exposition of ideas.
- Programming through problem-solving can be more engaging.
- Choice of programming language does matter.
- There ought to be a clearer distinction between teaching a language and teaching programming.

Conclusions

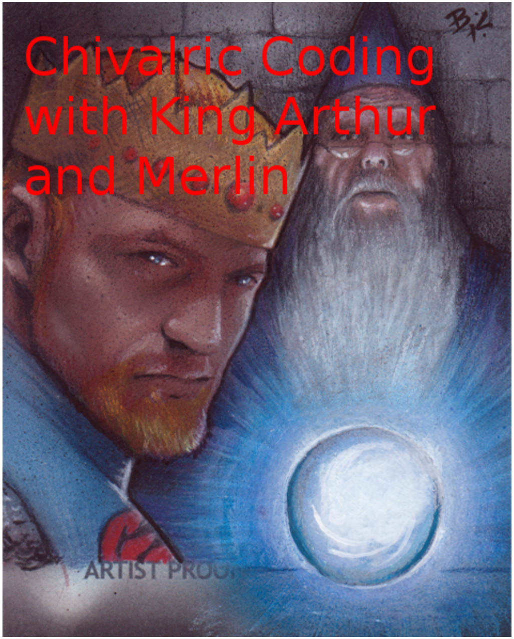
Are we getting better at writing programs that solve the given problem?

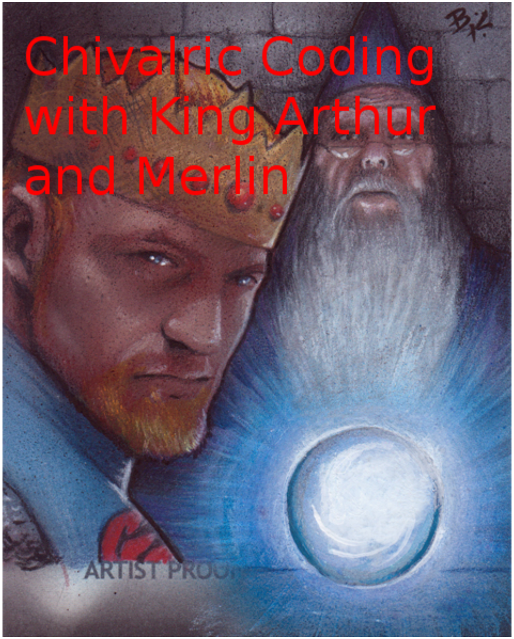
- The complexity of problems in introductory texts today would suggest so.

Is programming a specialist discipline, or will everyone in the future be a programmer?

- I claim the reverse: much of what would be done with "programming" in the past would be done with a spreadsheet or database today.

Chivalric Coding with King Arthur and Merlin





Thanks for listening. Comments and questions?