Programming Paradigms

Keith Mannock

Department of Computer Science and Information Systems
Birkbeck, University of London

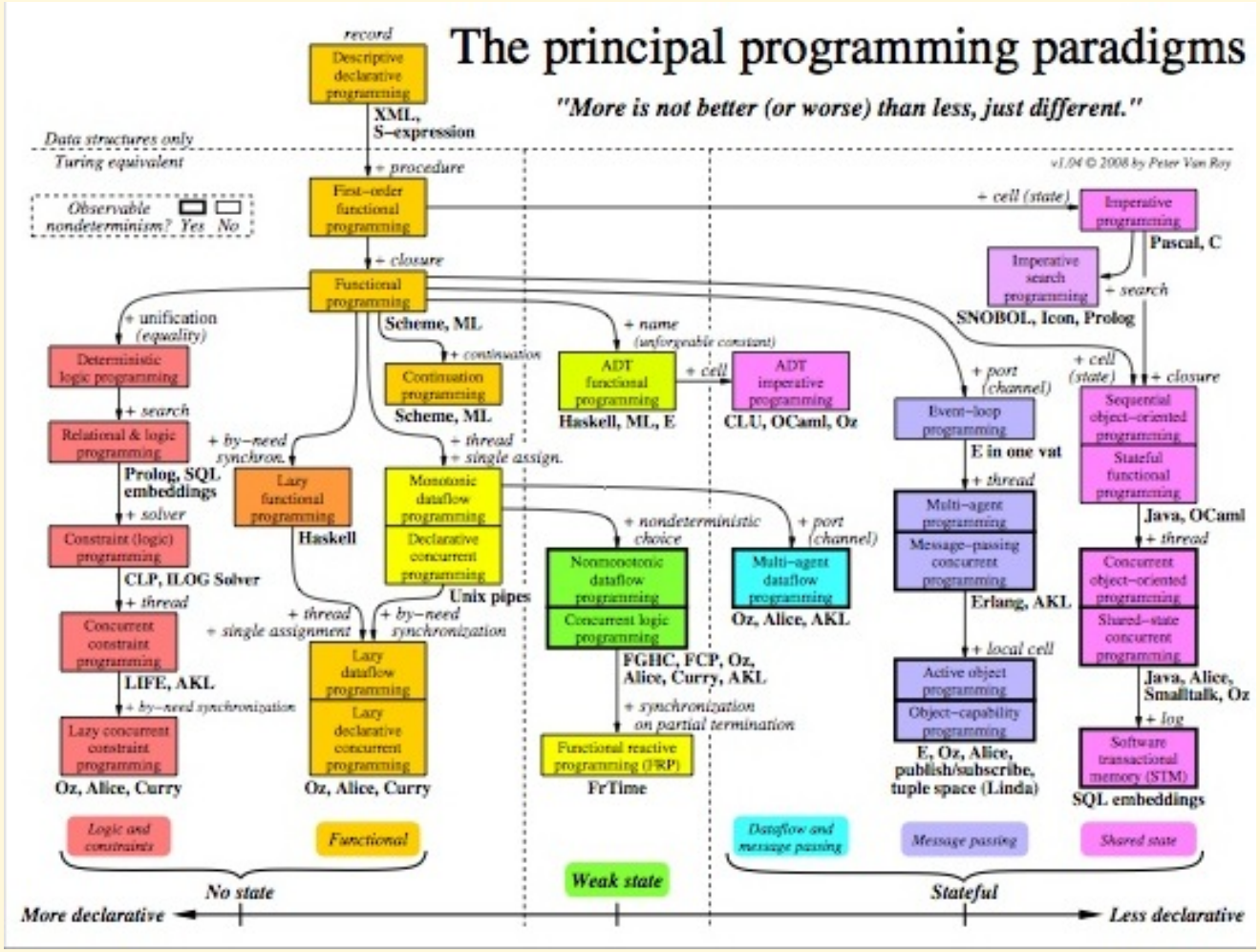
October 4, 2013

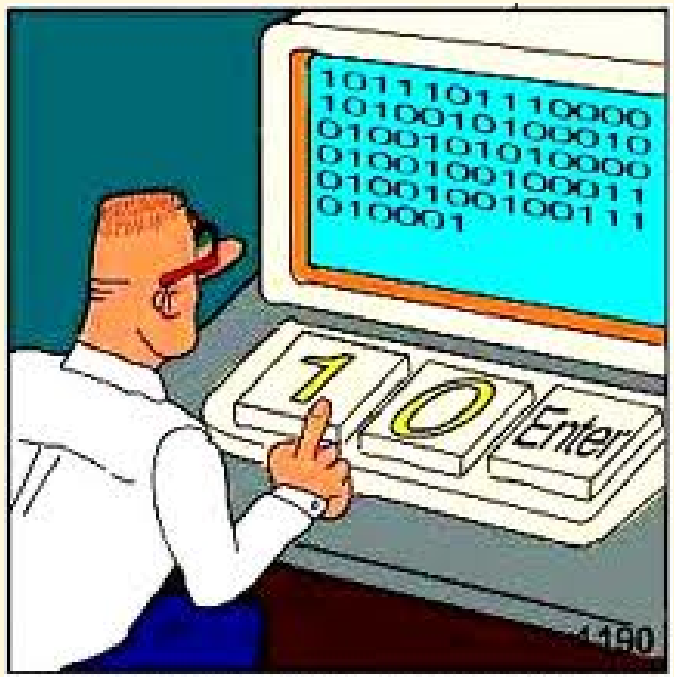
http://moodle.bbk.ac.uk

What is this course about?

- To understand the key differences between the various programming paradigms
- The applicability of these paradigms to different programming problems
- To deal with the changing requirements of the current day (and future) programmer
- To understand (so called) polyglot programming (the ability to mix and match languages and technologies appropriately) — which is fast becoming the norm.

Languages?


The principal programming paradigms
"More is not better (or worse) than less, just different."
v1.04 © 2008 by Peter Van Roy
Data structures only
Turing equivalent
Observable nondeterminism? Yes No
record
Descriptive declarative programming
XML, S-expression
+ procedure
First-order functional programming
+ closure
Functional programming
Scheme, ML
+ cell (state)
Imperative programming
Pascal, C
Imperative search programming
+ search
SNOBOL, Icon, Prolog
+ unification (equality)
Deterministic logic programming
+ search
Relational & logic programming
Prolog, SQL embeddings
+ solver
Constraint (logic) programming
CLP, ILOG Solver
+ thread
Concurrent constraint programming
LIFE, AKL
+ by-need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ by-need synchron.
Lazy functional programming
Haskell
+ thread
+ single assignment
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
+ continuation
Continuation programming
Scheme, ML
+ thread
+ single assign.
Monotonic dataflow programming
Declarative concurrent programming
Unix pipes
+ by-need synchronization
+ name (unforgeable constant)
ADT functional programming
Haskell, ML, E
+ cell
ADT imperative programming
CLU, OCaml, Oz
+ nondeterministic choice
Nonmonotonic dataflow programming
Concurrent logic programming
FGHC, FCP, Oz, Alice, Curry, AKL
+ synchronization on partial termination
Functional reactive programming (FRP)
FrTime
+ port (channel)
Multi-agent dataflow programming
Oz, Alice, AKL
+ port (channel)
Event-loop programming
E in one vat
+ thread
Multi-agent programming
Message-passing concurrent programming
Erlang, AKL
+ local cell
Active object programming
Object-capability programming
E, Oz, Alice, publish/subscribe, tuple space (Linda)
+ cell (state)
+ closure
Sequential object-oriented programming
Stateful functional programming
Java, OCaml
+ thread
Concurrent object-oriented programming
Shared-state concurrent programming
Java, Alice, Smalltalk, Oz
+ log
Software transactional memory (STM)
SQL embeddings
Logic and constraints
Functional
Dataflow and message passing
Message passing
Shared state
No state
Weak state
Stateful
More declarative
Less declarative



REAL Programmers code in BINARY.

Who are we?

Keith Mannock

- keith@dcs.bbk.ac.uk
- www.dcs.bbk.ac.uk/~keith
- x6713 MAL157

Trevor Fenner

- trevor@dcs.bbk.ac.uk
- www.dcs.bbk.ac.uk/~trevor
- x6704 MAL265

and possibly several others...

Paradigm I

What do we actually mean by the word paradigm?

"An example that serves as pattern or model."

The American Heritage Dictionary of the English Language

"A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalisations and the experiments performed in support of them are formulated"

The Merriam-Webster's Collegiate dictionary

Paradigm II

Programming paradigm (as used in this course)

- A pattern that serves as a *school of thoughts* for programming of computers

Programming technique

- Related to an algorithmic idea for solving a particular class of problems
  e.g., 'Divide and conquer' and 'program development by stepwise refinement'

Paradigm III

Programming style

- The way we express ourselves in a computer program
- Related to elegance or lack of elegance

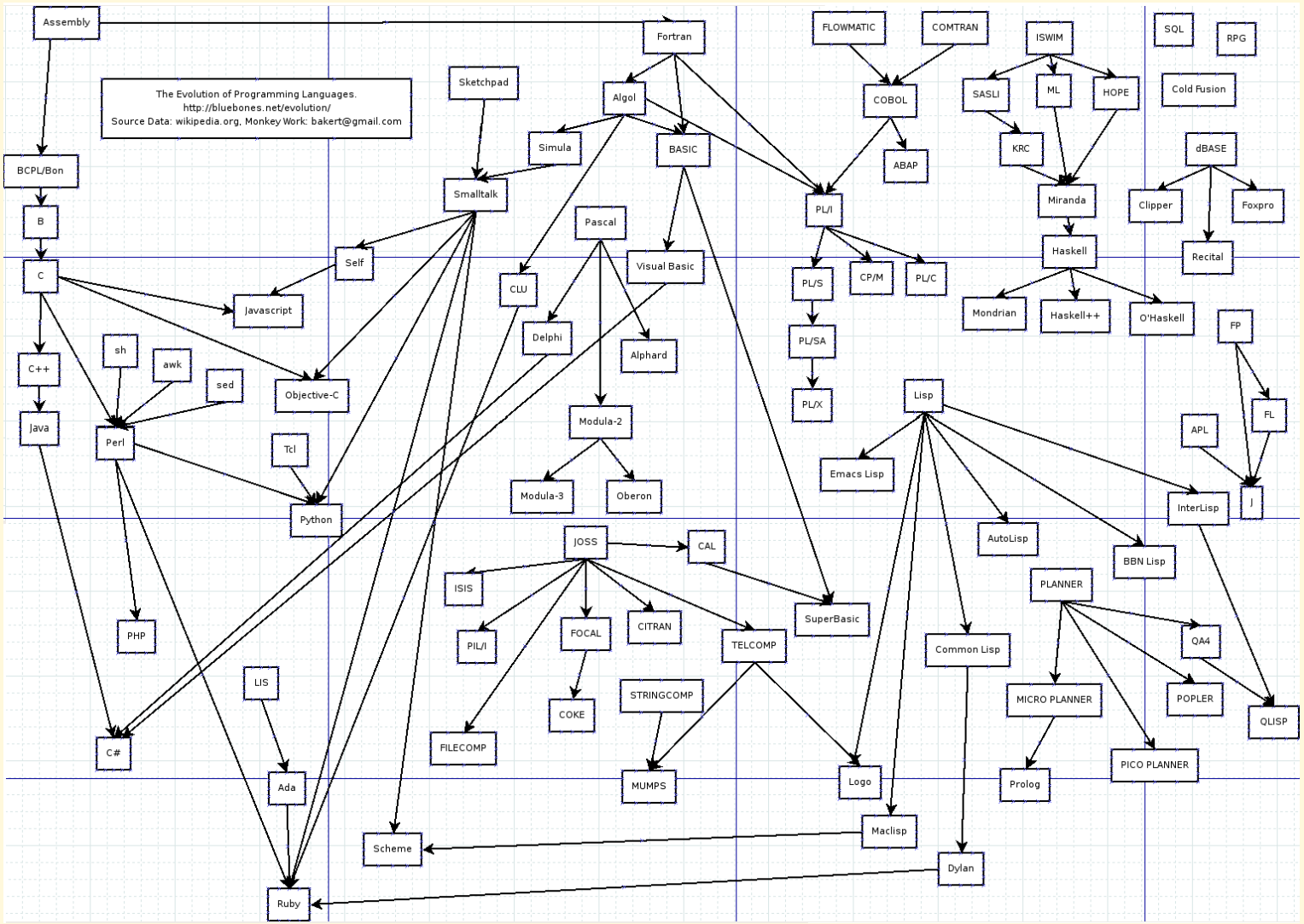
Programming culture

- The totality of programming behaviour, which often is tightly related to a family of programming languages
- The sum of a main paradigm, programming styles, and certain programming techniques

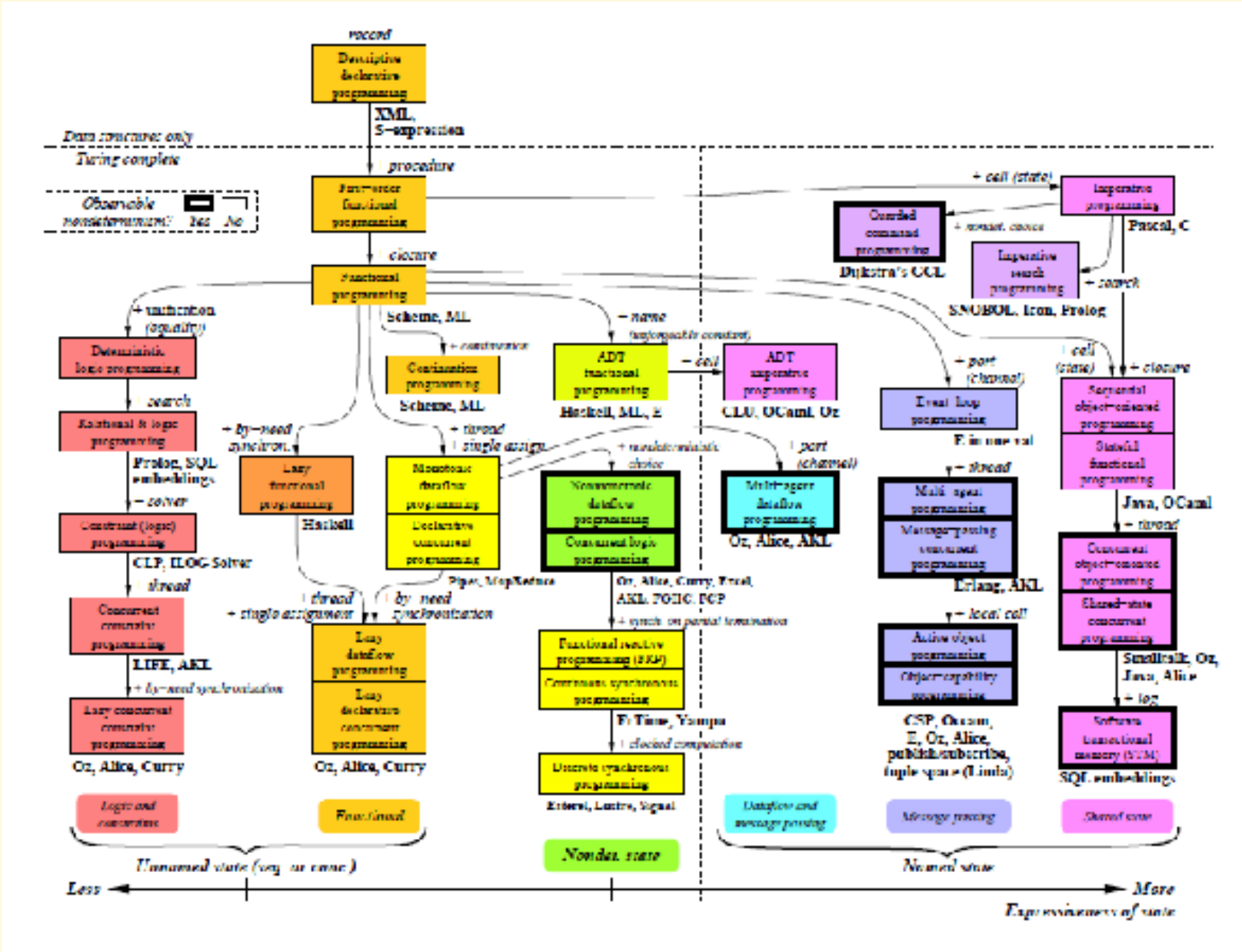
Programming Languages I

- There are many different programming languages out there
  - a couple of hundred, if not thousands...
- A quick look may reveal the following

Programming Languages II



Programming Languages III



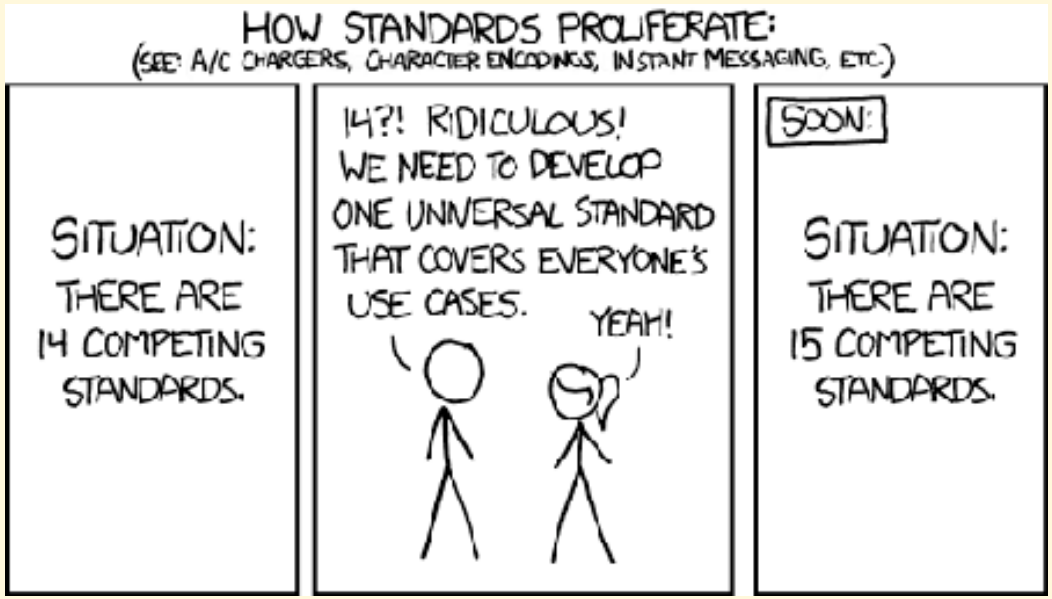
Huge Number of Languages I

Why are there so many different languages. . . ?

- "I wish there was a programming language that lets you do XYZ!"
- Some languages are highly specialised for certain domains
- "Let's create a language that combines all the neat features of languages X,Y, and Z!

. . . and this leads to so called *standards*

Huge Number of Languages II



Huge Number of Languages III

. . . and do we have to look at all of these languages?

- Fortunately not, there are more general underlying principles and concepts
- Programming languages can be categorised according to *programming paradigms*
  - We identify four main programming paradigms and a number of minor programming paradigms
  - A main programming paradigm stems from an idea within some basic discipline which is relevant for performing computations

Once you have understood these general concepts, it becomes easier to learn new programming languages

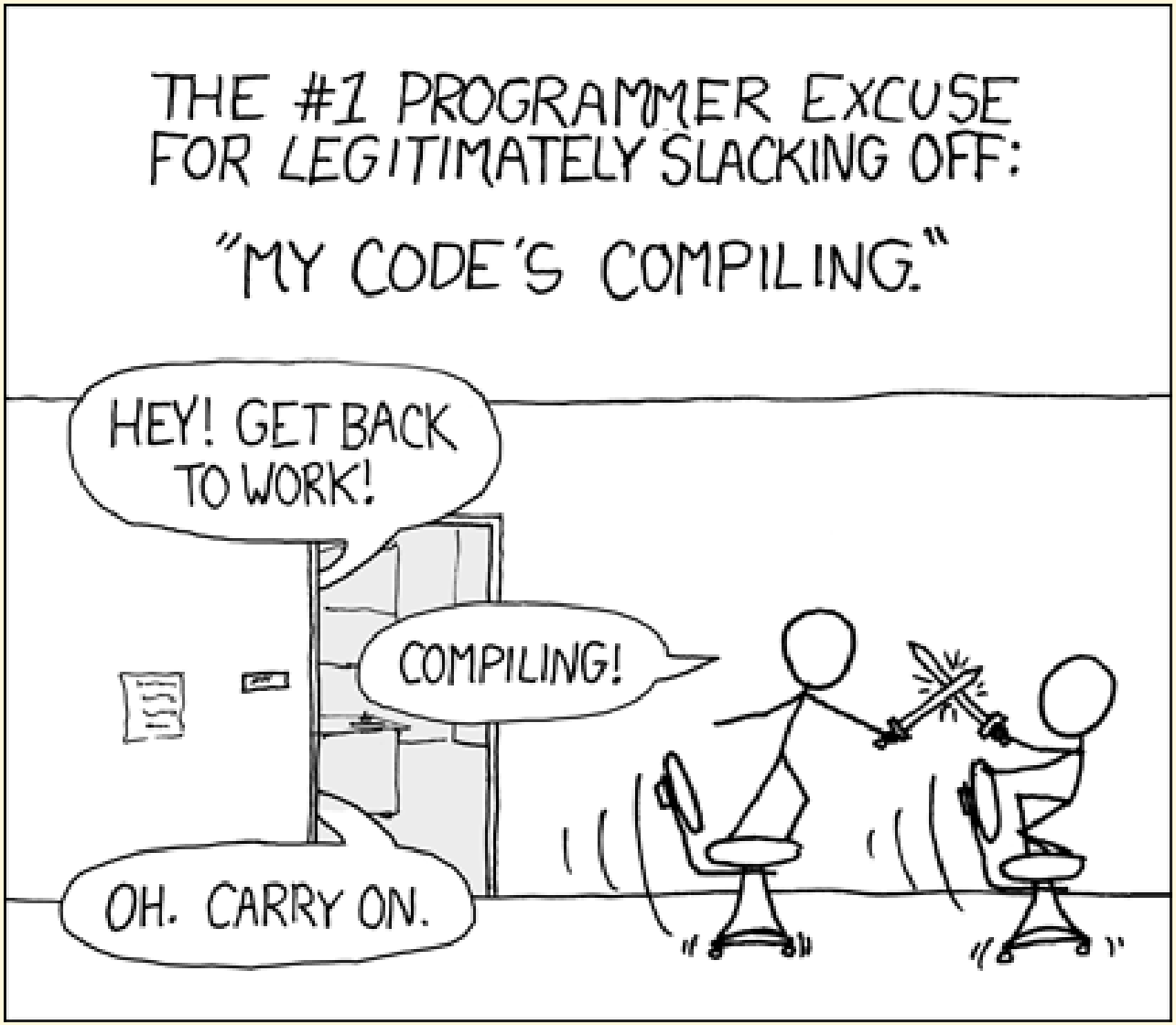
Huge Number of Languages IV

Main programming paradigms

- The imperative paradigm
- The functional paradigm
- The logical paradigm
- The object-oriented paradigm

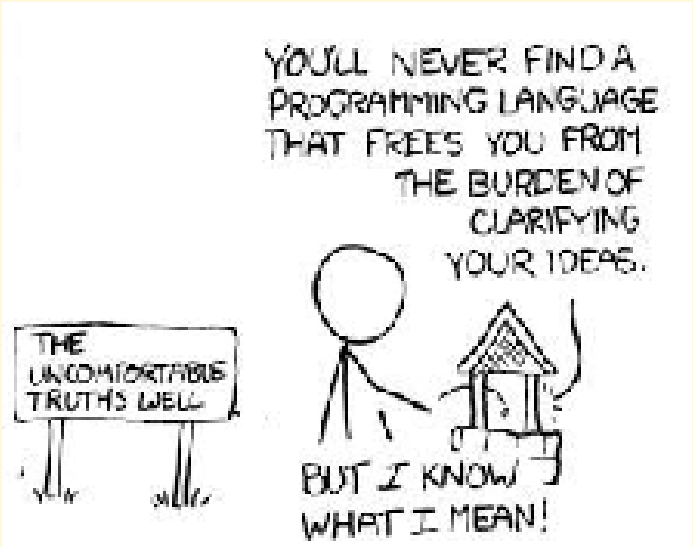
Some other possible programming paradigms

- The visual paradigm
- One of the parallel (concurrent) paradigms
- The constraint based paradigm
- ...


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Paradigms I

- However, does not mean that by just picking the right paradigm all problems vanish into thin air



Paradigms II

- Or put more elegantly:

  "There does not now, nor will there ever exist, a programming language in which it is the least bit hard to write bad programs."'

  L. Flon

Paradigms III

- In this course we are going to look at the most important paradigms
- We will also highlight strengths and weaknesses of each paradigm
- This will be done in a practical way using concrete languages:

> *"Learning to program is like learning to swim. No amount of theory is a substitute for diving into the pool."*
>
> *Joe Armstrong*

Paradigms IV

- Keeping to the swimming analogy: there are many different ways of swimming as well:
  - breaststroke
  - backstroke
  - crawl
  - butterfly
  - ...
- Each of them has certain advantages and disadvantages

Schedule

- Brief recap
  - Elements of programming languages
  - Imperative/procedural paradigm
- More paradigms and languages
  - Object-oriented: Ruby, Scala
  - Logic programming: Prolog
  - Functional: Clojure, Haskell
  - Concurrent: Erlang, Scala

Basic software requirements...

- The Java Development Kit (JDK) preferably version 1.8 (although 1.7 covers most things we will need)
- The documentation for the JDK
- The Integrated Development Environment (IDE) of your choice (although you can use the command line if you really want to)
- ... + the software for each language we will use!

Labs

The lab is scheduled for 8pm but we might "mix it up" a little...

- Check the Moodle website before you come, to see if the room has been moved or the schedule changed

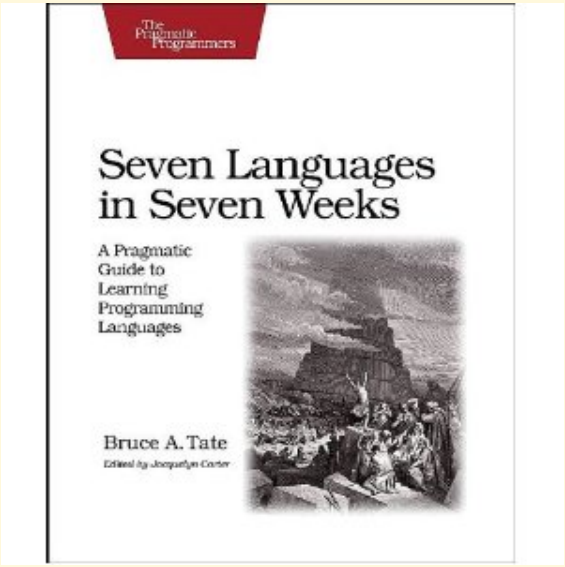
Everyone is expected to attend lab, but...

- If you understand the material on the worksheet, then you probably do not need to attend
- No new material will be presented in the lab sessions; some demos and tooling, but no new language features
- The "lab sheets" can be done at home — and probably will have to be completed there (or at least out of lab time)

Books/Literature

- The main book used for these lectures is

  Bruce A. Tate: Seven Languages in Seven Weeks, Pragmatic Bookshelf, 2012



- Other texts we reference in the course are listed on the website
- we will look at those now...

Assessment

Varies depending whether you are taking the UG or PG version of the course.

PG By two hour written examination (80%)

PG By practical coursework (20%) and exercises

UG By two hour written examination (75%)

UG By practical coursework (25%) and exercises

- Submission will be via Moodle
- The late policy is as stated in the degree handbook; the cutoff date is 14 days after the due date

How to get a good grade? I

- Start your assignments early!
  - This is the first and most important way to improve your grades
  - Programming takes a lot of time
  - Its not easy to predict how long a program will take
- Test your programs thoroughly
  - One or two simple tests are not enough
  - We often provide simple but incomplete tests, just to get you started
  - We will do thorough testing, even if you dont!

How to get a good grade? II

- Read the assignments carefully
- Do what is assigned, not *something like* what is assigned
- Learn to use your tools (e.g., eclipse, JUnit, etc.)
- Use comments and good style right from the beginning, not as a last-minute addition
- Review and understand the lectures

Who to ask when it all goes horribly wrong!

If you have questions about the course:

- Ask, email, or phone, the lecturer for that particular topics
- Ask or email Keith
- Ask or phone Trevor

Keith and email...

Feel free to send me email at:

`keith@dcs.bbk.ac.uk`

I get lots of spam, including several virus-carrying messages a week, so I run several filters.

To avoid my spam and virus filters:

- If your email concerns this course, please put ProgPara2013 somewhere in the subject line

General thoughts. . . I

On a personal note, my own experience has been that really understanding a programming paradigm is only possible one paradigm at a time and in languages which force you into the paradigm. Ideally, you would use a language which takes the paradigm to the extreme (hence the number of languages we will approach).

General thoughts. . . II

In multi-paradigm languages, it is much too easy to cheat and fall back on a paradigm that you are more comfortable with. Using a paradigm as a library is only really possible in languages like Scheme, or to a lesser extent, Scala, which are specifically designed for this kind of programming.

Learning lazy functional programming in Java, for example, is not a good idea, although there are libraries for that.

Questions thus far. . .