Advances in Programming Languages

APL1: What's so important about language?

Ian Stark

School of Informatics
The University of Edinburgh

Monday 12 January 2009
Semester 2 Week 1

What matters in a programming language?

Easy starter questions.

What matters in a programming language?

Easy starter questions.

- Name some programming languages.

What matters in a programming language?

Easy starter questions.

- Name some programming languages.
- Identify some of their features and characteristics.

What matters in a programming language?

Easy starter questions.

- Name some programming languages.
- Identify some of their features and characteristics.

We might like a language that is:

> Easy to learn, quick to write, expressive, concise, powerful, supported, well-provided with libraries, cheap, popular, ...

It might help us to write programs that are:

> Readable, correct, fast, reliable, predictable, maintainable, secure, robust, portable, testable, composable, ...

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

This claim is not without controversy; both in its original domain of linguistics, and as more recently applied to programming languages.

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Orwell: The purpose of Newspeak was not only to provide a medium of expression for the world-view and mental habits proper to the devotees of Ingsoc, but to make all other modes of thought impossible [*1984*, 1949]

Shaping the conceivable

Languages frame the way we think, and the programs we can imagine.

Sapir-Whorf Hypothesis

We dissect nature along lines laid down by our native language

Boole: Language is an instrument of human reason, not merely a medium for the expression of thought [An Investigation of the Laws of Thought, 1854]

Wittgenstein: The limits of my language mean the limits of my world [Tractatus Logico-Philosophicus, 1922]

Orwell: The purpose of Newspeak was not only to provide a medium of expression for the world-view and mental habits proper to the devotees of Ingsoc, but to make all other modes of thought impossible [*1984*, 1949]

Perlis: A language that doesn't affect the way you think about programming, is not worth knowing [Epigrams on Programming, 1982]

That's a bit philosophical

Does this really happen? Can programming languages help us write new kinds of program? Or just stop us from writing bad ones?

That's a bit philosophical

Does this really happen? Maybe.

- LISP S-expressions, metaprogramming, treating code as data.
- Higher-order functions. For example, *parser combinators*:

```
expr = (expr `then` opn `then` expr) `or` term
opn  = (char '+') `or` (char '-')
term = ...
```

- Laziness for infinite datastructures:

```
odds = 3 : [ x+2 | x<-odds ]  -- Self-referential list

pi = g(1,180,60,2) where      -- Gibbons's spigot
  g(q,r,t,i) =
     let (u,y)=(3*(3*i+1)*(3*i+2),div(q*(27*i-12)+5*r)(5*t))
     in y : g(10*q*i*(2*i-1),10*u*(q*(5*i-2)+r-y*t),t*u,i+1)
```

- [Your suggestion here...]

Programmability

One of the most significant feature of computers is the fact that they are *programmable*. Day-to-day, this is astoundingly underused.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Programmability

One of the most significant feature of computers is the fact that they are *programmable*. Day-to-day, this is astoundingly underused.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Turing writing about the Automatic Computing Engine ACE:

> Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a good deal of work of this kind to be done, ...

Programmability

One of the most significant feature of computers is the fact that they are *programmable*. Day-to-day, this is astoundingly underused.

Programmability means that computers can always do more. Best of all, you can program new ways to program.

Turing writing about the Automatic Computing Engine ACE:

> Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle-solving ability. There will probably be a good deal of work of this kind to be done, ...
>
> This process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.

[Proposed Electronic Calculator, 1945]

Abstraction

The concept of *abstraction* is an important route to the programmability of languages themselves.

Abstractions build upon each other: bytes, arrays, pointers, trees, files, sockets, objects, databases, procedures, functions, threads, behaviours, ...

Abstraction

The concept of *abstraction* is an important route to the programmability of languages themselves.

Abstractions build upon each other: bytes, arrays, pointers, trees, files, sockets, objects, databases, procedures, functions, threads, behaviours, ...

Abstraction frees up you to think about other things, and you should. Let the machine get on with its job.

Knuth: Premature optimization is the root of all evil

[Structured Programming with go to Statements, 1974]

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March
Week 11	Monday 23 March	Thursday 26 March

This gives 22 slots.

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots.

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots, including guest lectures, assignment & review tutorials.

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots, including guest lectures, assignment & review tutorials.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots, including guest lectures, assignment & review tutorials.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3;

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots, including guest lectures, assignment & review tutorials.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3; choice of topic must be made by the end of Week 4;

Time plan

Week 1	Monday 12 January	Thursday 15 January
Week 2	Monday 19 January	Thursday 22 January
Week 3	Monday 26 January	Thursday 29 January
Week 4	Monday 2 February	Thursday 5 February
Week 5	Monday 9 February	Thursday 12 February
Week 6	Monday 16 February	Thursday 14 February
Week 7	Monday 23 February	Thursday 21 February
Week 8	Monday 2 March	Thursday 5 March
Week 9	Monday 9 March	Thursday 12 March
Week 10	Monday 16 March	Thursday 19 March

This gives 20 slots, including guest lectures, assignment & review tutorials.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3; choice of topic must be made by the end of Week 4; report due by the end of Week 10.

Communication

Web

http://www.inf.ed.ac.uk/teaching/courses/apl/

The course web page carries lecture slides, a lecture log and links to resources mentioned, as well as occasional news and advice.

Lecturers

The most effective way to contact either lecturer is by personal email, from your University email address. However, many questions are even better posed on the course *newsgroup*.

Newsgroup

news://newsread.ed.ac.uk/eduni.inf.course.apl

You should read the course newsgroup regularly. It carries timely announcements about lectures, homework, and coursework. You can ask questions about the course, and respond to the questions of others.

See the course web page for information on how to access newsgroups.

What's out there?

Some example "advances in programming languages" for this course:

- Extensible records for typing objects in OCaml
- Specifying and statically checking behaviour of Java code
- LINQ and cross-language integration in .NET
- Patterns and abstractions for programming concurrent code
- Mobile code that carries its own proof of safety

In addition, the coursework will involve you finding out about a further topic, chosen from a similar list.

Crystal ball gazing

Some areas to watch, and possible drivers of future language design:

- Multicore
- Relaxed memory models
- Quantum computing
- General-purpose computing on GPUs, FPGAs
- {Cloud,distributed,mobile,web} computing
- Scripting
- Language-based security
- Multilanguage interoperability

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. Current long shot: synthetic biology and programming languages for life.

The Secret Agenda of the Functional Illuminati

All advances in the design of mainstream programming languages shall arise by transfer from existing functional languages.

Everything necessary can be found by contemplation of ML or Haskell.

The exceptionally adept may already discern all these in LISP.

- ✓ Automatic memory management (everywhere these days)
- ✓ Exceptions (ditto)
- ✓ Parametric polymorphism (see Java/C# generics)
- ✓ Implicit pointers (any OO language)
- ✓ First-class functions (C# delegates)
- ✓ Immutable values (see Java `string`)
- ✓ Closures (lambdas in C#, Visual Basic 9 (!), maybe Java 7?)
- ? Algebraic datatypes (still trying, but see Scala)
- ? ...

Homework

The next lecture is on Thursday, and concerns type systems and the language Objective Caml (OCaml). Before then, you should:

- Read the Chapter 1 of the Objective Caml manual, *The Core Language*, Sections 1.1–1.5.
- Read *A Hundred Lines of Caml*.
- Execute some of those lines on a convenient `ocaml` implementation.

Also, Wikipedia's *History of programming languages* article is an easy read and fairly informative.