Advances in Programming Languages

APL1: What's so important about language?

Ian Stark and David Aspinall

School of Informatics
The University of Edinburgh

Monday 11 January 2010
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 distinguishing characteristics.

What matters in a programming language?

Easy starter questions.

- Name some programming languages.
- Identify some of their distinguishing characteristics.

We might like a language that is:

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

What matters in a programming language?

Easy starter questions.

- Name some programming languages.
- Identify some of their distinguishing 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 : map (+2) odds
fibs = 1 : 1 : [ a+b | (a,b) <- zip fibs (tail fibs) ]

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...]

Properties

One of the defining feature of computers is that they are *programmable*.

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

Properties

One of the defining feature of computers is that they are *programmable*.

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, ...

Properties

One of the defining feature of computers is that they are *programmable*.

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]

Properties

One of the defining feature of computers is that they are *programmable*.

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]

That is:

> If you don't like the computer you have, you can create a better one

[Miller, LtU, 2009-05-11]

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, ...

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, ...

Programming an abstraction means that programmability itself can move up a level: to larger units and higher-order concepts.

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, ...

Programming an abstraction means that programmability itself can move up a level: to larger units and higher-order concepts.

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

Abstraction

The concept of *abstraction* adds significant power to programmability.

Abstractions build upon each other: bytes, strings, arrays, matrices, lists, maps, trees, pointers, files, sockets, objects, databases, instructions, procedures, functions, threads, agents, behaviours, ...

Programming an abstraction means that programmability itself can move up a level: to larger units and higher-order concepts.

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]

What's in the course?

The lectures will cover four sample areas of "advances in programming languages":

- Specifying and statically checking behaviour of Java code
- Type classes in Haskell can do anything
- Patterns and abstractions for programming concurrent code
- LINQ and cross-language integration in .NET

Lectures also specify reading and exercises on the topics covered. This homework is not assessed, but it is essential in order to fully participate in the course.

There is also substantial piece of written coursework which contributes 20% of students' course grade. This requires investigation of a topic in programming languages and writing a 10-page report with example code.

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March
Week 11	Monday 22 March	Thursday 25 March

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March
Week 11	Monday 22 March	Thursday 25 March

This gives 22 slots.

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

This gives 20 slots,

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

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

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

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

20% of the course grade is for a written investigation of a novel language feature, from a list provided, with your own working code examples.

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

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

20% of the course grade is for a written investigation of a novel language feature, from a list provided, with your own working code examples.

The topic list will be presented in the next lecture;

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

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

20% of the course grade is for a written investigation of a novel language feature, from a list provided, with your own working code examples.

The topic list will be presented in the next lecture; intermediate report due by the end of Week 5;

Time plan

Week 1	Monday 11 January	Thursday 13 January
Week 2	Monday 18 January	Thursday 20 January
Week 3	Monday 25 January	Thursday 28 January
Week 4	Monday 1 February	Thursday 4 February
Week 5	Monday 8 February	Thursday 11 February
Week 6	Monday 15 February	Thursday 13 February
Week 7	Monday 22 February	Thursday 20 February
Week 8	Monday 1 March	Thursday 4 March
Week 9	Monday 8 March	Thursday 11 March
Week 10	Monday 15 March	Thursday 18 March

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

20% of the course grade is for a written investigation of a novel language feature, from a list provided, with your own working code examples.

The topic list will be presented in the next lecture; intermediate report due by the end of Week 5; final report due by the end of Week 8.

Communication

Web

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

The course web page gives basic information, and through the semester will carry lecture slides, details of coursework and exams.

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 through comments on the course blog.

The mailing list apl-students@inf.ed.ac.uk reaches all APL students and staff. Check http://lists.inf.ed.ac.uk/ to see that you are listed correctly.

Blog

http://blob.inf.ed.ac.uk/aplcourse/

You should read the course blog. It carries the lecture log, slides, information about further reading and background material expanding on the references in the lectures.

You can add comments, and respond to the questions of others. Please do.

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
- Warehouse-scale computing and upwards
- {Cloud,mobile,web} computing
- Dynamic languages
- Language-based security and assurance

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. What would you put on your 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
- Warehouse-scale computing and upwards
- {Cloud,mobile,web} computing
- Dynamic languages
- Language-based security and assurance

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. What would you put on your list? What's next?

Synthetic biology and programming languages for life? see Venter's *Synthia*

The Secret Agenda of the Functional Illuminati

All advances in the design of mainstream programming languages shall arise 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 C, Java 7?)
- ? Algebraic datatypes (still trying, but see Scala)
- ? First-class continuations (. . .)

Homework

The next lecture is 9am on Thursday. Before then:

1. Read the Wikipedia article on *History of programming languages*. (If you find it's missing something, fix that.)

2. What is a *closure*? How does it differ from a *function pointer*? What are closures good for? What languages have them?

   Post one or more of the following as comments on the blog entry for this lecture:

   - An explanation of closures, or a pointer to an explanation online.
   - An example of a closure in use.
   - A language that has closures, and how you would represent one of the examples in it.

   Please don't duplicate each other's answers — and no more than one language each, leave some for others.