Advances in Programming Languages

APL1: What's so important about language?

Ian Stark

School of Informatics
The University of Edinburgh

Thursday 10 January 2008
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 and abstraction

The single 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 and abstraction

The single 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.

The concept of *abstraction* is an important way this appears in programming languages.

Abstractions build upon each other: bytes, arrays, pointers, lists, trees, files, sockets, databases, objects, procedures, 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]

What's out there?

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

- Type-safe extensible records in OCaml
- Join-patterns for concurrency in Polyphonic C#
- LINQ and cross-language integration in .NET
- JML for specification and checking in Java
- Cyclone for pointer management in C

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
- Quantum computing
- Non-von-Neumann architectures, FPGAs
- {Cloud,distributed,mobile,web} computing
- Scripting
- Security
- Evidence-based trust
- Language interoperability
- OS programming

Don't take this too seriously: some of these have been on the "soon to be hot" list for decades. Today's 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# and Visual Basic 9 (no, really!))
- ? Algebraic datatypes (still trying)
- ? ... to be continued ...

Type taster

Java has strong static typing: all programs are checked for type-safety at compile-time. Bytecode is checked again on loading, before execution.

Java also has subtyping: a value of one type may be used at any more general type. So `String < Object`, and every `String` is an `Object`.

Not all is well with Java types

```
String[] a = { "Hello" };  // A small string array
Object[] b = a;            // Now a and b are the same array
b[0] = Boolean.FALSE;      // Drop in a Boolean object
String s = a[0];           // Oh, dear
System.out.println(s);     // This isn't going to be pretty
```

This compiles without error or warning: in Java, if `S<T` then `S[] < T[]`. Except that it isn't. So every array assignment gets a runtime check.

Exercise

The next lecture is on Monday, and concerns Objective Caml (OCaml). Before then, you should:

- Read the Chapter 1 of the Objective Caml manual, *The Core Language*.
- 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.