Why Programming Languages Matter

Andrew P. Black

Portland State University
Portland, Oregon
Why Programming Languages Matter to me and a bunch of other People

Andrew P. Black
Portland State University
Portland, Oregon
Win a Turing Award!
Win a Turing Award!
Turing Awards related to PL

1. Backus, John (1977)
3. Iverson, Ken (1979)
5. Lamport, Leslie (2013)
7. Milner, Robin (1991)
But they missed ...

1. Backus, John (1977)
3. Iverson, Ken (1979)
5. Lamport, Leslie (2013)
7. Milner, Robin (1991)
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Backus, John</td>
<td>1977</td>
</tr>
<tr>
<td>2</td>
<td>Hoare, Tony</td>
<td>1980</td>
</tr>
<tr>
<td>3</td>
<td>Iverson, Ken</td>
<td>1979</td>
</tr>
<tr>
<td>4</td>
<td>Kay, Alan</td>
<td>2003</td>
</tr>
<tr>
<td>5</td>
<td>Lamport, Leslie</td>
<td>2013</td>
</tr>
<tr>
<td>6</td>
<td>Liskov, Barbara</td>
<td>2008</td>
</tr>
<tr>
<td>7</td>
<td>Milner, Robin</td>
<td>1991</td>
</tr>
<tr>
<td>8</td>
<td>Naur, Peter</td>
<td>2005</td>
</tr>
<tr>
<td>9</td>
<td>Wirth, Niklaus</td>
<td>1984</td>
</tr>
<tr>
<td>10</td>
<td>Allen, Fran</td>
<td>2006</td>
</tr>
<tr>
<td>11</td>
<td>Dahl, Ole-Johan</td>
<td>2001</td>
</tr>
<tr>
<td>12</td>
<td>Dijkstra, Edsger</td>
<td>1972</td>
</tr>
<tr>
<td>13</td>
<td>Floyd, Bob</td>
<td>1978</td>
</tr>
<tr>
<td>14</td>
<td>McCarthy, John</td>
<td>1971</td>
</tr>
<tr>
<td>15</td>
<td>Nygaard, Kristen</td>
<td>2001</td>
</tr>
<tr>
<td>16</td>
<td>Perlis, Alan</td>
<td>1966</td>
</tr>
<tr>
<td>17</td>
<td>Ritchie, Dennis M.</td>
<td>1983</td>
</tr>
<tr>
<td>18</td>
<td>Scott, Dana</td>
<td>1976</td>
</tr>
</tbody>
</table>

But they missed …

Friday, 30 October 2015
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Backus, John</td>
<td>1977</td>
</tr>
<tr>
<td>2</td>
<td>Hoare, Tony</td>
<td>1980</td>
</tr>
<tr>
<td>3</td>
<td>Iverson, Ken</td>
<td>1979</td>
</tr>
<tr>
<td>4</td>
<td>Kay, Alan</td>
<td>2003</td>
</tr>
<tr>
<td>5</td>
<td>Lamport, Leslie</td>
<td>2013</td>
</tr>
<tr>
<td>6</td>
<td>Liskov, Barbara</td>
<td>2008</td>
</tr>
<tr>
<td>7</td>
<td>Milner, Robin</td>
<td>1991</td>
</tr>
<tr>
<td>8</td>
<td>Naur, Peter</td>
<td>2005</td>
</tr>
<tr>
<td>9</td>
<td>Wirth, Niklaus</td>
<td>1984</td>
</tr>
<tr>
<td>10</td>
<td>Allen, Fran</td>
<td>2006</td>
</tr>
<tr>
<td>11</td>
<td>Dahl, Ole-Johan</td>
<td>2001</td>
</tr>
<tr>
<td>12</td>
<td>Dijkstra, Edsger*</td>
<td>1972</td>
</tr>
<tr>
<td>13</td>
<td>Floyd, Bob*</td>
<td>1978</td>
</tr>
<tr>
<td>14</td>
<td>McCarthy, John*</td>
<td>1971</td>
</tr>
<tr>
<td>15</td>
<td>Nygaard, Kristen*</td>
<td>2001</td>
</tr>
<tr>
<td>16</td>
<td>Perlis, Alan*</td>
<td>1966</td>
</tr>
<tr>
<td>17</td>
<td>Ritchie, Dennis M.*</td>
<td>1983</td>
</tr>
<tr>
<td>18</td>
<td>Scott, Dana</td>
<td>1976</td>
</tr>
</tbody>
</table>
Win a Turing Award!

Analysis of Algorithms
Artificial Intelligence
Computational Complexity

Cryptography

Data Structures  Databases  Education  Error Correcting Codes  Finite Automata  Graphics
Interactive Computing  Internet Communications  List Processing  Numerical Analysis
Numerical Methods  Object Oriented Programming  Operating Systems  Personal Computing
Program Verification  Programming

Programming Languages

Proof Construction  Software Theory  Software Engineering

Verification of Hardware and Software Models  Computer Systems  Machine Learning
Parallel Computation
Win a Turing Award!

Analysis of Algorithms  Artificial Intelligence  Combinatorial Algorithms
Compilers
Computational Complexity  Computer Architecture  Computer Hardware
Cryptography

Data Structures  Databases  Education  Error Correcting Codes  Finite Automata  Graphics
Interactive Computing  Internet Communications  List Processing  Numerical Analysis
Numerical Methods  Object Oriented Programming  Operating Systems  Personal Computing
Program Verification  Programming

Programming Languages  Proof Construction  Software Engineering

Software Theory  Verification of Hardware and Software Models  Computer Systems  Machine Learning
Parallel Computation

Portland State University

Friday, 30 October 2015
My Personal Journey

• 1977–1981: Graduate student, Oxford
• 1981–1986: Assistant Professor, Washington
• 1986–1994: Engineer & Researcher, Digital
• 1994–1999: Department Head, OGI
• 2000–2004: Professor, OGI
• 2004+: Professor, Portland State
My Personal Journey

• 1977–1981: Graduate student, Oxford
  1978: IBM TJ Watson Research Center
• 1981–1986: Assistant Professor, Washington
• 1986–1994: Engineer & Researcher, Digital
• 1994–1999: Department Head, OGI
• 2000–2004: Professor, OGI
• 2004–: Professor, Portland State
My Personal Journey

• 1977–1981: Graduate student, Oxford
  1978: IBM TJ Watson Research Center

• 1981–1986: Assistant Professor, Washington

• 1986–1994: Engineer & Researcher, Digital
  1998: Xerox PARC

• 1994–1999: Department Head, OGI

• 2000–2004: Professor, OGI

• 2004–: Professor, Portland State
My Personal Journey

- **1977–1981:** Graduate student, Oxford
  
  1978: IBM TJ Watson Research Center

- **1981–1986:** Assistant Professor, Washington

- **1986–1994:** Engineer & Researcher, Digital

- **1994–1999:** Department Head, OGI
  
  1998: Xerox PARC

- **2000–2004:** Professor, OGI
  
  2001: University of Bern  2002: Intel

- **2004–:** Professor, Portland State
My Personal Journey

• 1977–1981: Graduate student, Oxford
  1978: IBM TJ Watson Research Center
• 1981–1986: Assistant Professor, Washington
• 1986–1994: Engineer & Researcher, Digital
• 1994–1999: Department Head, OGI
  1998: Xerox PARC
• 2000–2004: Professor, OGI
  2001: University of Bern  2002: Intel
• 2004–: Professor, Portland State
  2011: Microsoft  2002: Edinburgh
Programming is Hard

I want to make it easier
1978–80: 3R

• “Reading, ‘riteing, and ‘rithmetic”

• Programming language designed for readability
  - Names made up of multiple words
  - Block (procedure) names can have arguments, e.g. delete [i]th line of page[p]

• Flat (no nesting): Blocks and Blocklets
  - No loops, No defaults
4.5. Scanning One Word

This block scans the current line and returns the next word or perhaps a null string if one is not found. A word is a letter followed by zero or more letters, digits, or underscore characters.

LET New Word := Get One Word BE
USES Current Character
RESULT New Word IS TEXT
INVARIABLE Underscore Character IS '_'
New Word := '
Remove Front Blanks
IF (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'Z')
  New Word := New Word + Current Character
Get Next Character
Add Characters Until Delimiter
IF NOT (...  
  (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'Z'))
  PASS
OTHERWISE CHAOS

WHERE Add Characters Until Delimiter IS
IF (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'A') OR ...
  (Current Character = Underscore Character)
New Word := New Word + Current Character
Get Next Character
Add Characters Until Delimiter
IF NOT ( ...
  (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'A') OR ...
USES Current Character
RESULT New Word IS TEXT
INVARIABLE Underscore Character IS '_'
New Word := ''
Remove Front Blanks
IF (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'Z')
New Word := New Word + Current Character
Get Next Character
Add Characters Until Delimiter
IF NOT (...
  (Current Character >= 'a' AND Current Character <= 'z') OR ...
  (Current Character >= 'A' AND Current Character <= 'Z'))
PASS
OTHERWISE CHAOS

WHERE Add Characters Until Delimiter IS
  IF (Current Character >= 'a' AND Current Character <= 'z') OR ...
    (Current Character >= 'A' AND Current Character <= 'A') OR ...
    (Current Character >= '0' AND Current Character <= '9') OR ...
    (Current Character = Underscore Character)
  New Word := New Word + Current Character
  Get Next Character
  Add Characters Until Delimiter
  IF NOT (...
    (Current Character >= 'a' AND Current Character <= 'z') OR ...
    (Current Character >= 'A' AND Current Character <= 'Z') OR ...
    (Current Character >= '0' AND Current Character <= '9') OR ...
    (Current Character = Underscore Character))
  PASS
  OTHERWISE CHAOS
END OF BLOCK { new word := get one word }
Influences

- Algol 60
- Cobol?
- Hoare Triples, Dijkstra’s predicate transformers
- Top-down design
- A year at IBM
- Brian Shearing
  - knew that he needed a language
Reflections

“The concept of a program consisting of English text interspersed with 3R was easily grasped, but its use was more difficult than I anticipated. The main problem ... is a feeling of duplicating in the English what’s I’ve already coded in 3R ... The code specifies the details in a concise and comprehensible manner, [and] in a superior style.”

Howard Matsuoka
Language as a Simplifier
Language as a Simplifier
Language as a Simplifier

- Programming in Smalltalk is *also* a life-changing experience
Language as a Simplifier

• Programming in Smalltalk is *also* a life-changing experience

• Once you understand how freeing it is get rid of the junk, you will never want to go back
Eden Programming Language

• Eden Project (1980–1984) — early attempt to build a “distributed, integrated” computing system.

• EPL provided:
  - concurrency inside Eden objects
  - synchronous (local or remote) object invocation
  - capabilities
  - strings

• Implemented by translating to Concurrent Euclid
Sending an Invocation

Client Process

RootDirectory.Lookup(
    UserName, LoginDirectory, Status)

Call

and Return

Directory Stub Procedure

procedure Lookup(...)  
{Pack arguments into ESCII}

Dispatcher.SynchInvoke(...)  
{unpack results}

dend

Invocation
Receiving an Invocation

Directory Code

```
invocation procedure Lookup(...) =
begin
  {code to search the directory
   and find the requested key}
end Lookup
```

Client Process

```
loop
  Dispatcher.ReceiveOperations(I, EnquiryOps)
  CallInvocationProcedure(I)
end loop
```

CIP for Directory

```
procedure CallInvocationProcedure
  (I: InvocationHandle) =
begin
  {Fetch values from ESCII.
   if OperationName is "Lookup" then }
  Directory.Lookup( ... )
  {Pack results into reply ESCII}
  Dispatcher.ReplyMsg(I, Results)
```
Reflections

• Eden saw itself as *distributed systems* research
  - no one on the project knew that they needed a programming language!
• In hindsight: EPL was essential
• Partly language, partly kit of components
1983–87: Emerald

• Follow-on to EPL, but a “Real” Programming Language
  - Hides implementation choices that EPL revealed
  - Efficient (as in C) implementation
The People

Andrew Black

Eric Jul

Norm Hutchinson

Henry (Hank) Levy

Friday, 30 October 2015
The People

Andrew Black
Norm Hutchinson
Eric Jul (Hank) Levy

Friday, 30 October 2015
1983–87: Emerald

• Background:
  - Eric Jul (Simula 67, Concurrent Pascal),
  - Norm Hutchinson (Simula),
  - Hank Levy (Capability architectures, system-building at Digital)

• Addressed building a distributed system as a language problem

• Emerald separated “semantics” from “locatics”
Emerald Features

- Object constructors
- Concurrency
- Failure handling
- Parameterized types
- Location-independent invocation
- Compiled code about as efficient as C
const initialObject ← object initialObject
  const limit ← 10

const newobj ← monitor object innerObject
  var flip : Boolean ← true  % true => print hi next
  const c : Condition ← Condition.create

export operation Hi
  if ! flip then
    wait c
  end if
  stdout.PutString[“Hi
”]
  flip ← false
  signal c
end hi

export operation Ho
  if flip then
    wait c
  end if
  stdout.PutString[“Ho
”]
  flip ← true
  signal c
end ho

initially
  stdout.PutString[“Starting Hi Ho program
”]
end initially
end innerObject

const hoer ← object hoer
  process
    var i : Integer ← 0
    loop
      exit when i = limit
      newobj.Hi
      i ← i + 1
    end loop
  end process
end hoer

process
  var i : Integer ← 0
  loop
    exit when i = limit
    newobj.Ho
    i ← i + 1
  end loop
end process
end initialObject

const hoer ← object hoer
  process
    var i : Integer ← 0
    loop
      exit when i = limit
      newobj.Hi
      i ← i + 1
    end loop
  end process
end hoer

process
  var i : Integer ← 0
  loop
    exit when i = limit
    newobj.Ho
    i ← i + 1
  end loop
end process
end initialObject
Reflections

• About 20 years before its time
  - NSF called it “unimplementable”
  - Still generating PhDs in 2006
SOSP Referee’s didn’t agree...

#90 "Fine-Grained Mobility in the Emerald System"

Referee's Report

This is a straightforward implementation of a simple idea. It is hard to see what is unique about this operating system.
Reflections

• About 20 years before its time
  - NSF called it “unimplementable”
  - Still generating PhDs in 2006
• Not widely used, but widely influential
  - ANSA DPL, OMG CORBA, INRIA’s *Guide*, Birrell et al.’s Network Objects, the ANSI Smalltalk standard
• We were our own customers. We realized that we needed a language ...
2001–present: Traits

• a language feature, not a language
• a Trait is a Smalltalk class without any slots
• traits can be
  - combined with +,
  - modified with @ (alias) and – (exclusion)
  - used in other traits and classes.
• **Trait** = set of methods, without instance vars

• **Sum, alias, exclude and uses** as combinator

---

The diagrams illustrate the three composition operations involving traits. The ellipses depict the operations; the fat arrows show their inputs and outputs. The open arrow represents subclassing. The notation a represents a method with name a and body m.

The **sum** operation takes two traits T and U as input; the result is a trait T+U that contains the union of all of the non-confliting methods. Where T and U conflict (e.g., at c), the resulting method is an explicit conflict marker.

The **overriding** operation combines some explicit definitions with an existing trait. In the figure, the explicit definitions of methods b and c override those obtained from the trait T, and the definition of d is added. The resulting trait V contains no conflicts because the definition of b has been overridden.

The **inheritance** operation is used to create a new subclass D from an existing superclass C, an existing trait T, and some new, explicitly given definitions. Explicit definitions (e.g., of b) override those obtained from the trait; definitions in the trait (e.g., of a) ovveride those obtained from the superclass.
Influences

• Deep experience with Smalltalk
• The sad history of multiple inheritance
  “multiple inheritance is good, but there is no good way to do it”
  
  Steve Cook channeling Alan Snyder
• Nathanael Schärli, who cut the gordian knot
• A little lattice theory
• Excellent toolbuilding environment & skills
Reflections

• *Smallest* contribution
• Largest impact?
  - Pearl 6, Java, Pharo, Visualworks, Fortress, Racket, Ruby, C#, Scala, Joose, PHP, ActionScript, ...
• We underestimated the importance of programming tools
  - many of the properties we claimed for traits depended also on tool support
2010–present: Grace

- Simple O-O language for teaching
  - block-structured
  - dialects.
  - optional, gradual types
  - indentation matters
- An effort at *consolidation*, not *innovation*
- Open-source implementation
Linked List

```java
method with(*a) {
    def result = empty
    a.do { each -> result.add(each) }
    return result
}

class empty {
    class node(d, n) {
        var data is public := d
        var next is public := n
        method asString { "{data}|{next}" }
        method insert(value) {
            next := node(value, next)
        }
    }

    def null = Singleton.named "null"
    def top = node("header", null)
    var lastNode := top
```
method size {
  // returns the number of elements in self
  var result := 0
  var current := top
  while { current.next ≠ null } do {
    current := current.next
    result := result + 1
  }
  return result
}

method do(action:Block1) {
  // applies action to each element of self
  var current := top
  while { current.next ≠ null } do {
    current := current.next
    action.apply(current.data)
  }
}

method search(needle) ifAbsent(action) {
  // searches for needle in self. Returns the first node
  // containing needle if it is found; otherwise, applies action.
  var current := top
  while { current.next ≠ null } do {

Influences

- Teaching with inappropriate languages
  - Java: mixes paradigms, verbose, complex
  - Python: stupid defaults, objects are an afterthought
  - Smalltalk: no types, no interfaces
Reflections

• The *consumer* is a novice student
  - but the *customer* is an instructor in a introductory programming course

• Surprisingly challenging to please both
  - e.g., clean object model *or* existing practice?

• Design skills ⇔ implementation skills

• [http://www.gracelang.org](http://www.gracelang.org)
Meta-Reflections

• I’ve had a lot of fun over the last 35 years
  - Maybe I’ve also had some impact
• But programming is still too hard
• The (recent) focus on Programming Languages rather than Programming Systems hasn’t helped
  - less science and more engineering?
What keeps me coming back?

• I like *fixing things*
  - there’s plenty to fix in programming!

• Programming languages are an *enabler*
  - for others (3R, EPL)
  - for programmers (Traits)
  - for students (*Grace*)

• Programming languages are about communication
  - still refining my writing and communication skills
  - in English, and in program
Why is progress so slow?

- Programming languages are central to everything that we build
  - You would be crazy to build a 100 kloc system with an untested language.

- Tooling and libraries are as important, or more important, than the language
  - they take time to build and evolve
Why else?
Why else?

• A programming language is not just a means for programmers to communicate with computers
Why else?

• A programming language is not just a means for programmers to communicate with computers

• It is also a means for programmers to communicate with programmers —
Why else?

- A programming language is not just a means for programmers to communicate with *computers*
- It is also a means for programmers to communicate with *programmers* —
- It is a *social*, as well as a *technical*, enabler
Why else?

• A programming language is not just a means for programmers to communicate with *computers*

• It is also a means for programmers to communicate with *programmers* —

• It is a *social*, as well as a *technical*, enabler
  - social change is slow
Why else?

- A programming language is not just a means for programmers to communicate with computers
- It is also a means for programmers to communicate with programmers —
- It is a social, as well as a technical, enabler
  - social change is slow
  - but enjoys the “100th monkey” effect
What about others?
What about others?

A quick survey of the members of IFIP WG 2.16 on language design ...
What about others?

A quick survey of the members of IFIP WG 2.16 on language design ... 

... revealed a lot of passion
Creating

“The power to create out of pure thought”

Jonathan Edwards

“A universal tool”

“In the beginning was the word”

Cristina Lopes
Magic

Programmers are like wizards ... except that the magic is real!

PLs are “spell systems”

Sean McDirmid

“Any sufficiently-advanced technology is indistinguishable from magic”

Arthur C. Clarke
Foundational

- Software is the most important infrastructure for ... basically everything
- Software is totally dependent on programming languages
- Programming languages are the most important infrastructure for writing software ... and thus for anything and everything!

James Noble
Fun

Building things is fun!

Building things *that build things* is doubleplus fun!

Jonathan Aldrich
Are we there yet?
Are we there yet?  No!
Since Fortran, people have been saying that we don’t need new languages.

Yet, languages continue to evolve ... and few of us would want to go back to Fortran.

Roberto Ierusalimschy
Language as “Law Enforcement”
Language as “Law Enforcement”
Language as “Law Enforcement”
Language as “Law Enforcement”

“Law” of Physics
Language as “Law Enforcement”
Language as “Law Enforcement”

The value of a language can be in what it prevents you from doing

Hence: libraries are not the answer
Language as “Law Enforcement”

The value of a language can be in what it prevents you from doing

Hence: libraries are not the answer

- No library is ever going to ensure that there are no race conditions in my Java program
Languages shape thought
Languages shape thought

Whorfianism, or “Linguistic Relativity”
Languages shape thought

Whorfianism, or “Linguistic Relativity”
Learning a new language “changes the path of least resistance”

Tom van Cutsem
Languages shape thought

Whorfianism, or “Linguistic Relativity”
Learning a new language “changes the path of least resistance”

Tom van Cutsem
Languages shape thought
Languages shape thought
Languages shape thought

“You can't trust the opinions of others, because of the Blub paradox: they’re satisfied with whatever language they happen to use, because it dictates the way they think about programs.”

Paul Graham
Languages shape thought
Languages shape thought
Languages shape thought

“A language that doesn’t affect the way you think about programming, is not worth knowing”

Alan Perlis
Languages shape thought
Languages shape thought

My Recommendation:
Languages shape thought

My Recommendation:

- *Do* program in a pure functional language
Languages shape thought

My Recommendation:

- *Do* program in a pure functional language
- *Do* program with pure objects (Smalltalk)
Languages shape thought

My Recommendation:

- *Do* program in a pure functional language
- *Do* program with pure objects (Smalltalk)
- *Do* program with CSP
Languages shape thought

My Recommendation:

- *Do* program in a pure functional language
- *Do* program with pure objects (Smalltalk)
- *Do* program with CSP
- *Do* try Logic Programming (but not Prolog!)
Languages shape thought

My Recommendation:

- *Do* program in a pure functional language
- *Do* program with pure objects (Smalltalk)
- *Do* program with CSP
- *Do* try Logic Programming (but not Prolog!)

Use them for a serious project
PL Reading List

1. *Notation as a tool of thought*. Iverson

2. *Programming as Theory-building*. Naur

3. *Beating the Averages*. Graham (and commentary thereon at c2.org)

4. *The Development of the Emerald Programming Language*. Black *et al.* HoPL III


8. *Babel-17*. Delany