Agile Software Development

MSc in Computer Science

Produced by

Eamonn de Leastar

edeleastar@wit.ie

Department of Computing, Maths & Physics

Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit

Programming Langages
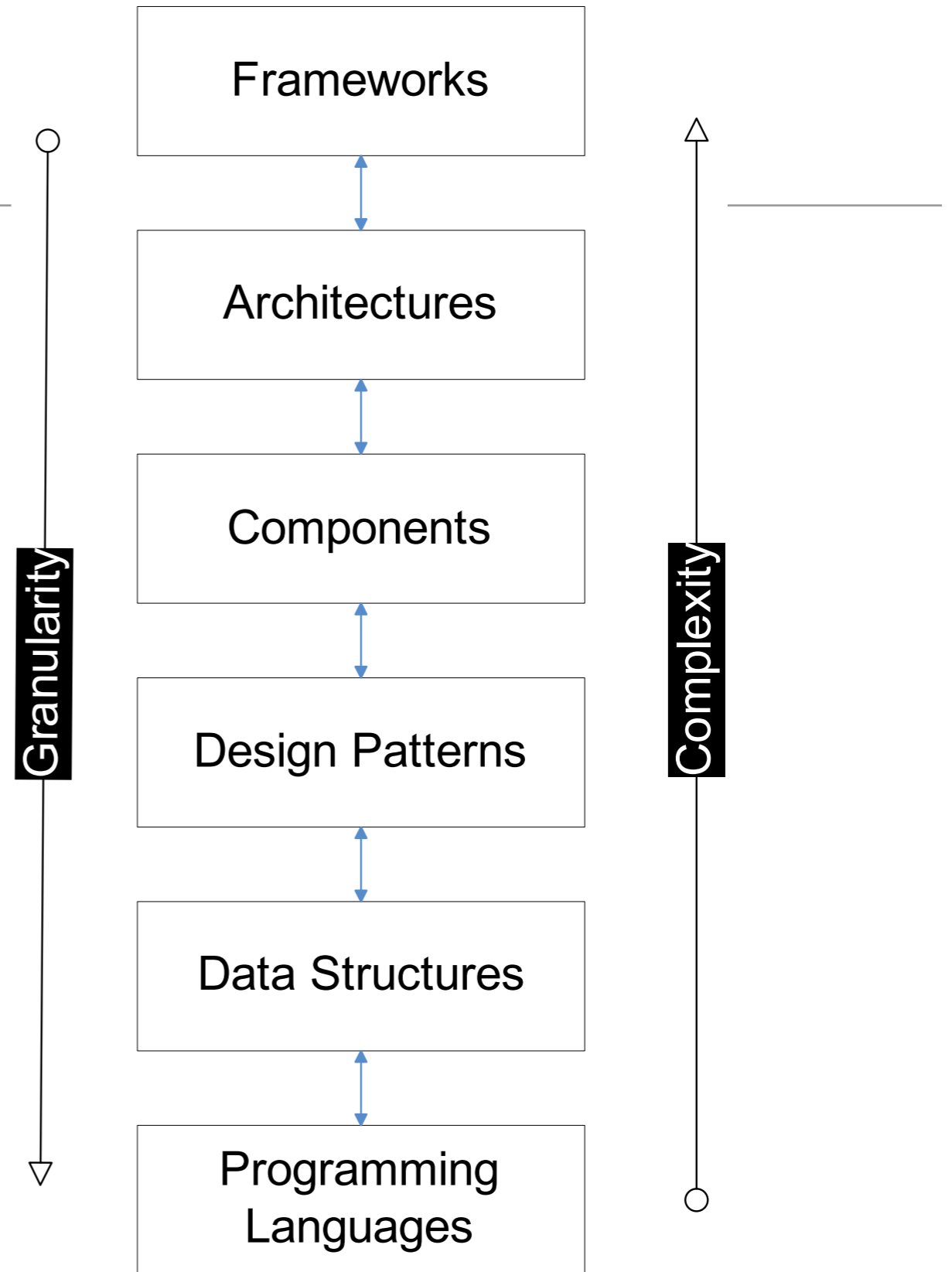
Agile Software Development

Programming Languages

- Context

- Family Trees

- Characteristics

- Typing Spectrum

- Static vs Dynamic Typing Example

- Other considerations..

Context

- Choice of programming language can have profound effect on accessibly of knowledge at higher layers

Granularity

Complexity

| Frameworks |
| Architectures |
| Components |
| Design Patterns |
| Data Structures |
| Programming Languages |

Family Trees

Mother Tongues

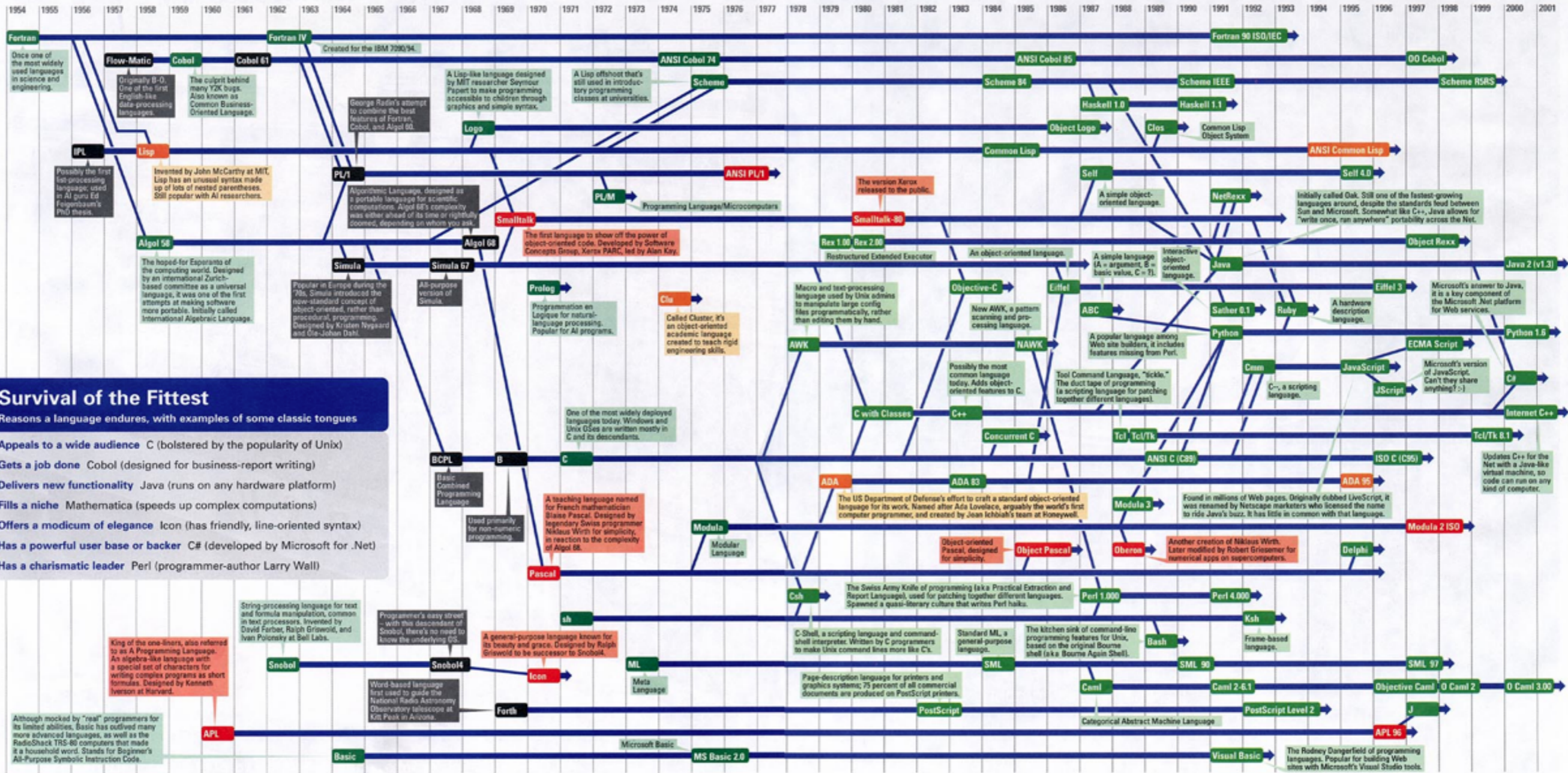**Tracing the roots of computer languages through the ages**

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at www.informatik.uni-freiburg.de/Java/misc/lang_list.html. – Michael Menduno

Key

1954	Year Introduced
▮ (green)	**Active:** thousands of users
▮ (orange)	**Protected:** taught at universities; compilers available
▮ (red)	**Endangered:** usage dropping off
▮ (black)	**Extinct:** no known active users or up-to-date compilers
→ (blue)	Lineage continues

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Timeline years: 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001

Languages and annotations shown on the family tree:

Fortran — Once one of the most widely used languages in science and engineering. → **Fortran IV** (Created for the IBM 7090/94.) → **Fortran 90 ISO/IEC**

Flow-Matic → **Cobol** (Originally B-0. One of the first English-like data-processing languages.) → **Cobol 61** (The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.) → **ANSI Cobol 74** → **ANSI Cobol 85** → **OO Cobol**

IPL — Possibly the first list-processing language; used in AI guru Ed Feigenbaum's PhD thesis. → **Lisp** (Invented by John McCarthy at MIT, Lisp has an unusual syntax made up of lots of nested parentheses. Still popular with AI researchers.) → **Common Lisp** → **ANSI Common Lisp**

A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax. → **Logo**

A Lisp offshoot that's still used in introductory programming classes at universities. → **Scheme** → **Scheme 84** → **Scheme IEEE** → **Scheme R5RS**

Haskell 1.0 → **Haskell 1.1**

Object Logo → **Clos** — Common Lisp Object System

George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60. → **PL/1** (Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask.) → **ANSI PL/1**

PL/M — Programming Language/Microcomputers

Self (A simple object-oriented language.) → **Self 4.0** — The version Xerox released to the public.

NetRexx — Initially called Oak. Still one of the fastest-growing languages around, despite the standards feud between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.

Algol 58 (The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.) → **Algol 68**

Simula (Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Designed by Kristen Nygaard and Ole-Johan Dahl.) → **Simula 67** (All-purpose version of Simula.)

Smalltalk (The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.) → **Smalltalk-80**

Rex 1.00 Rex 2.00 — Restructured Extended Executor → An object-oriented language. → **Object Rexx**

A simple language (A = argument, B = basic value, C = ?). → **Java** (A simple object-oriented language.) (Interactive object-oriented language.) → **Java 2 (v1.3)**

Prolog — Programmation en Logique for natural-language processing. Popular for AI programs.

Cls — Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.

AWK (Macro and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.) → **NAWK**

Objective-C — New AWK, a pattern scanning and processing language. Adds object-oriented features to C.

Eiffel → **Eiffel 3** — Microsoft's answer to Java, it is a key component of the Microsoft .Net platform for Web services.

ABC (A popular language among Web site builders, it includes features missing from Perl.) → **Python** → **Python 1.6**

Sather 0.1 → **Ruby** — A hardware description language.

Cmm — C--, a scripting language. → **ECMA Script**

JavaScript — Microsoft's version of JavaScript. Can't they share anything? :-) → **JScript** → **C#**

Tcl Tcl/Tk — Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages). → **Tcl/Tk 8.1**

C with Classes → **C++** → **Internet C++**

Concurrent C → **ANSI C (C89)** → **ISO C (C95)** — Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.

BCPL — Basic Combined Programming Language → **B** (Used primarily for non-numeric programming.) → **C**

Pascal — A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.

ADA (The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team at Honeywell.) → **ADA 83** → **ADA 95**

Module → **Modula** (Modular Language) → **Modula 3** — Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language. → **Modula 2 ISO**

Object-oriented Pascal, designed for simplicity. → **Object Pascal** → **Oberon** (Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.) → **Delphi**

Csh → **Perl** (The Swiss Army knife of programming (aka Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku.) → **Perl 1.000** → **Perl 4.000**

sh → C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's. → **Ksh**

String-processing language for text and formula manipulation, common in text processors. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs. → **Snobol** → **Snobol4**

Programmer's easy street – with this descendant of Snobol, there's no need to know the underlying OS.

A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4. → **Icon**

King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard. → **APL** → **APL 96**

ML — Meta Language → Standard ML, a general-purpose language. → **SML** → **SML 90** → **SML 97**

Caml → **Caml 2-6.1** → **Objective Caml** → **O Caml 2** → **O Caml 3.00**

Page-description language for printers and graphics systems; 75 percent of all commercial documents are produced with PostScript. → **PostScript** → **PostScript Level 2**

Bash — Frame-based language. (C-shell... programming features for Unix, based on the original Bourne shell (aka Bourne Again Shell).)

Categorical Abstract Machine Language → **J**

Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizona. → **Forth**

Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Beginner's All-Purpose Symbolic Instruction Code. → **Basic** → **Microsoft Basic** → **MS Basic 2.0** → **Visual Basic** — The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft's Visual Studio tools.
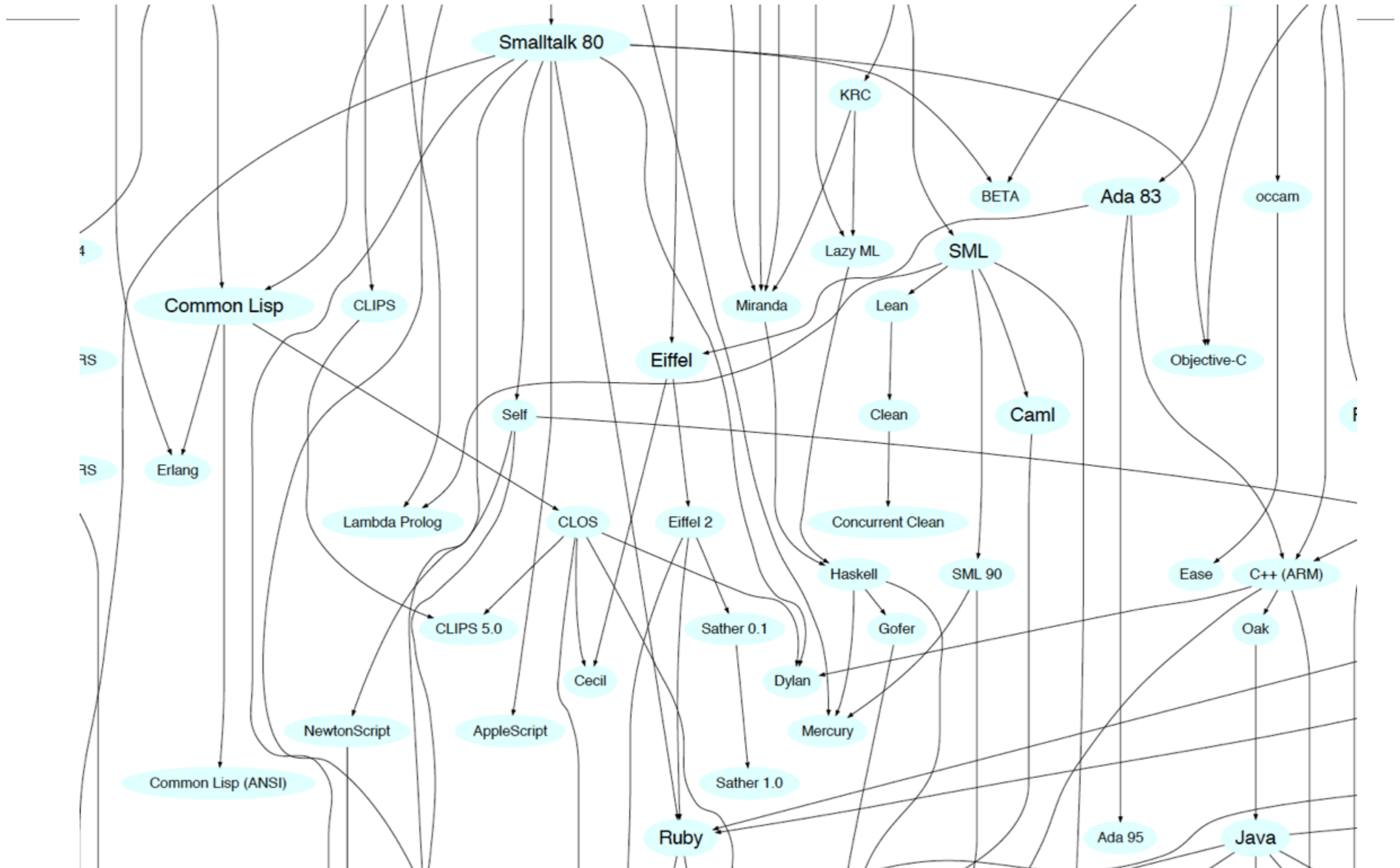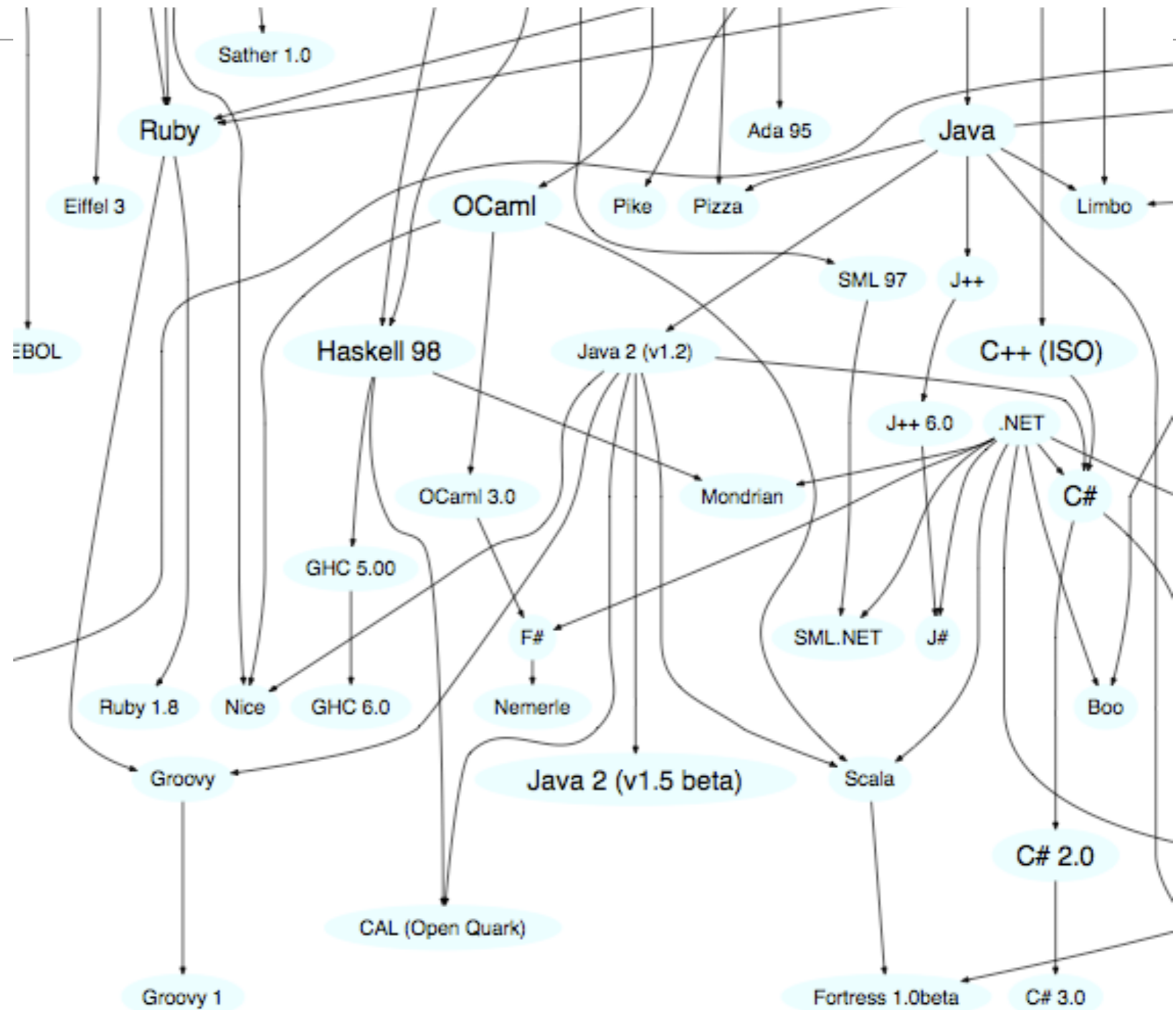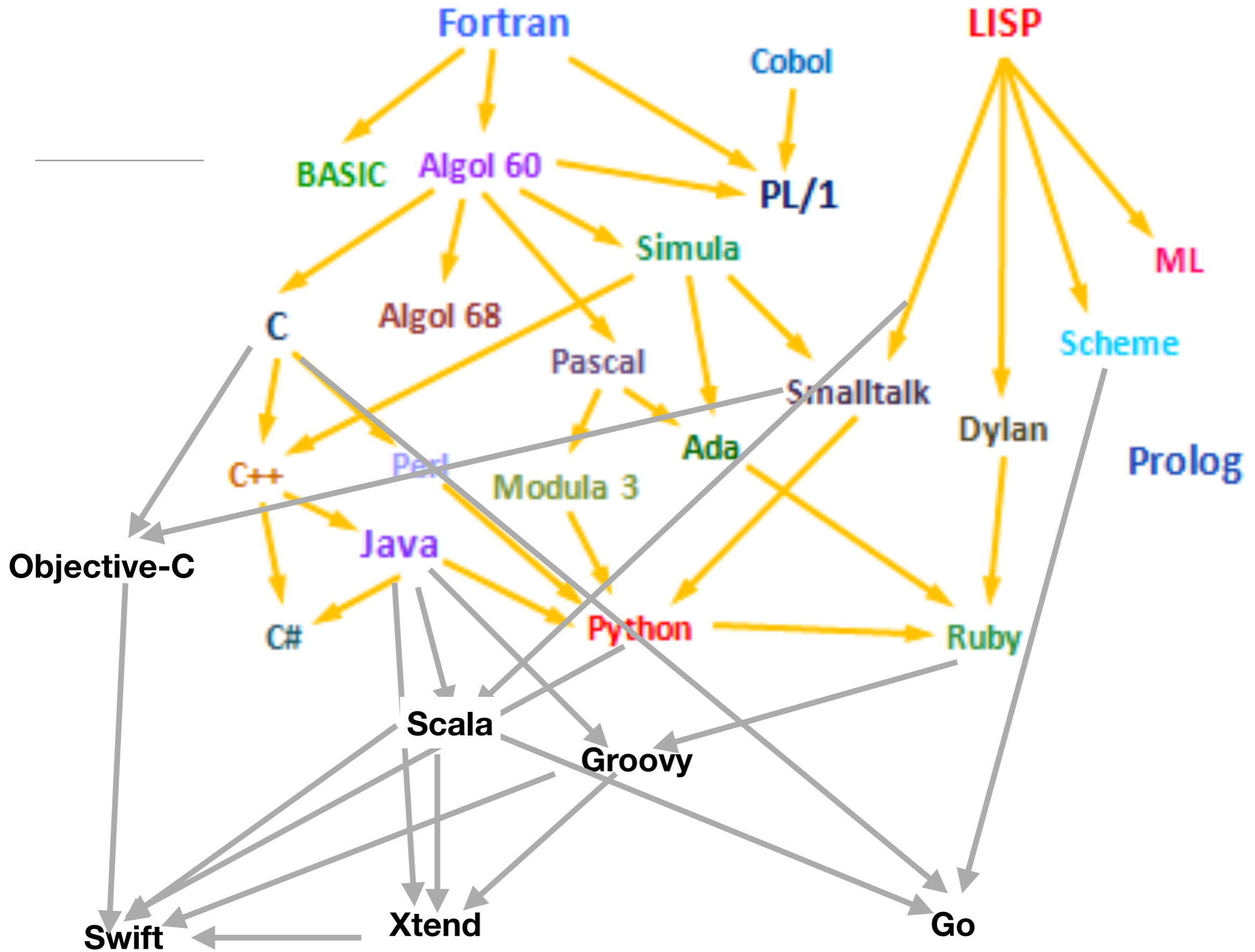
Family Tree (2)

Family Tree (3)

Ruby, Groovy, Java, Scala Cluster

Programming Languages as Reactions

(from Kevin Kelleher).

- *Algol:* Assembly language is too low-level.

- *Pascal:* Algol doesn't have enough data types.

- *Modula:* Pascal is too wimpy for systems programming.

- *Simula:* Algol isn't good enough at simulations.

- *Smalltalk*: Not everything in Simula is an object.

- *Fortran:* Assembly language is too low-level.

- *Cobol*: Fortran is scary.

- *PL/1:* Fortran doesn't have enough data types.

- *Ada:* Every existing language is missing something.

...Continued

- *Basic:* Fortran is scary.

- *APL:* Fortran isn't good enough at manipulating arrays.

- *J:* APL requires its own character set.

- *C:* Assembly language is too low-level.

- *C++:* C is too low-level.

- *Java:* C++ is a kludge. And Microsoft is going to crush us.

- *C#:* Java is controlled by Sun

...Continued

- *Lisp:* Turing Machines are an awkward way to describe computation.

- *Scheme*: MacLisp is a kludge.

- *T:* Scheme has no libraries.

- *Common Lisp:* There are too many dialects of Lisp.

- *Dylan*: Scheme has no libraries, and Lisp syntax is scary.

- *Perl:* Shell scripts/awk/sed are not enough like programming languages.

- *Python*: Perl is a kludge.

- *Ruby:* Perl is a kludge, and Lisp syntax is scary.

- *Groovy:* Ruby isn't compatible with Java

- *Scala*: Haskell is scary

Characteristics

Paul Grahams Wish List for a Programming Language

http://www.paulgraham.com/diff.html

1. Conditionals

2. A function type

3. Recursion

4. Dynamic typing

5. Garbage collection

6. Programs composed of expressions

7. A symbol type

8. A notation for code using symbols and trees

9. The whole language there all the time

Lisp programming Language has all of these features (since mid 1960's)

Java?

1.Conditionals

2.A function type (but coming in Java 8)

3.Recursion

4.Dynamic typing

5.Garbage collection

6.Programs composed of expressions

7.A symbol type

8.A notation for code using symbols and trees

9.The whole language there all the time

Groovy/Ruby/Python/Scala/Xtend
(from Neal Ford)

1. Conditionals

2. A function type

3. Recursion

4. Dynamic typing (or Type Inference)

5. Garbage collection

6. Programs composed of expressions

7. A symbol type

8. A notation for code using symbols and trees

9. The whole language there all the time

+ Metaprogramming

Groovy/Ruby/Python/Scala
(from Neal Ford)

1. Conditionals

2. A function type

3. Recursion

4. Dynamic typing (or Type Inference)

5. Garbage collection

6. Programs composed of expressions

7. A symbol type

8. A notation for code using symbols and trees

9. The whole language there all the time

+ Metaprogramming

Typing

Dynamic ⟷ Static

Amount of type checking enforced by the
compiler vs. leaving it to the runtime

Strong ←——————————————————→ Weak

How the runtime constraints you from treating objects of different types (in other words treating memory as blobs or specific data types)

http://blogs.agilefaqs.com/2011/07/11/dynamic-typing-is-not-weak-typing/

Typing Spectrum

Dynamic

- Python
- Ruby

- Smalltalk
- Groovy

- Javascript
- PHP

- C
- C++

- Objective-C

- Java
- C#

Static

Strong

Weak

Static to Dynamic

Java Example

- Java algorithm to filter a list of strings

- Only printing those shorter than 3 (in this test case).

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 1

- Also a valid
  Groovy program...

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 1

- Do we need generics?

- What about semicolons...

- Should standard libraries be imported?

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Groovy 2

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(Liststrings, length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Groovy 2

- Do we need the static types?

- Must we always have a main method and class definition?

- Consistency (size or length)?

```
class Erase
{
  public static void main(String[] args)
  {
    List names = new ArrayList()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    Erase e = new Erase()
    List short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  public List filterLongerThan(Liststrings, length)
  {
    List result = new ArrayList();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Groovy 3

```groovy
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

Groovy 3

- Should we have a special notation for lists?

- And special facilities for list processing?

```
def filterLongerThan(strings, length)
{
  List result = new ArrayList();
  for (String s : strings)
  {
    if (s.length() < length + 1)
    {
      result.add(s)
    }
  }
  return result
}

List names = new ArrayList()
names.add("Ted")
names.add("Fred")
names.add("Jed")
names.add("Ned")
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
for (String s : short_names)
{
  System.out.println(s)
}
```

Groovy 4

```groovy
def filterLongerThan(strings, length)
{
    return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

Groovy 4

- Method needed any longer?

- Is there an easier way to use common methods (e.g. println)?

- Are brackets always needed?

```
def filterLongerThan(strings, length)
{
   return strings.findAll {it.size() <= length}
}

names = ["Ted", "Fred", "Jed", "Ned"]
System.out.println(names)
List short_names = filterLongerThan(names, 3)
System.out.println(short_names.size())
short_names.each {System.out.println(it)}
```

Groovy 5

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

```groovy
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

Java vs Groovy?

Static to Inferred

Another Approach to Types?

- *Type Inference* : the compiler draws conclusions about the types of variables based on how programmers use those variables.

  - Yields programs that have some of the conciseness of Dynamically Typed Languages

  - But - decision made at *compile time*, not at *run time*

  - More information for static analysis - refactoring tools, complexity analysis. bug checking etc...

- Haskell, Scala, **Xtend**

```
object InferenceTest1 extends Application
{
  val x = 1 + 2 * 3          // the type of x is Int
  val y = x.toString()       // the type of y is String
  def succ(x: Int) = x + 1   // method succ returns Int values
}
```

Typing Spectrum

Dynamic

- Python   - Smalltalk   - Javascript
- Ruby   - Groovy   - PHP

Inferred

- Xtend
- Scala
- Go
- Swift

- C
- C++

- Java
- C#   - Objective-C

Static

Strong   *Weak* 39

Java Example
-> **XTend**

- Unlike Groovy - this is NOT an XTend Program

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

def & var

Are semicolons necessary?

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    var Erase e = new Erase();
    var List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

No semicolons

Can some types be inferred?

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var List<String> names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var Erase e = new Erase()
    var List<String> short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (String s : short_names)
    {
      System.out.println(s)
    }
  }

  def List<String> filterLongerThan(List<String> strings, int length)
  {
    var List<String> result = new ArrayList<String>()
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Type inference

What about Collection Literals?

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = new ArrayList<String>()
    names.add("Ted")
    names.add("Fred")
    names.add("Jed")
    names.add("Ned")
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Collection Literals

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    for (s : short_names)
    {
      System.out.println(s)
    }
  }

  def filterLongerThan(List<String> strings, int length)
  {
    var result = new ArrayList<String>()
    for (s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s)
      }
    }
    return result
  }
}
```

Can Lambas simplify code?

Lambdas

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)
    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
    val result = new ArrayList<String>()
    strings.forEach[ if (it.length() < length + 1)
      {
        result.add(it)
      }]
    result
  }
}
```

What are List Comprehensions?

Filters/List Comprehensions

```
import java.util.List;

class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    System.out.println(names)
    var e = new Erase()
    var  short_names = e.filterLongerThan(names, 3)

    System.out.println(short_names.size())
    short_names.forEach[System.out.println(it)]
  }

  def filterLongerThan(List<String> strings, int length)
  {
    val list = strings.filter[it.length() <= 3]
    list
  }
}
```

Do we need the class Erase at all?

Final Version

```
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

java

```
class Erase
{
  def static void main(String[] args)
  {
    var names = #["Ted", "Fred", "Jed", "Ned"]
    println(names)
    var  short_names =  names.filter[it.length() <= 3]
    println(short_names.size())
    short_names.forEach[println(it)]
  }
}
```

xtend

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
println short_names.size()
short_names.each {println it}
```

groovy

48

Java Example

- Java algorithm to filter a list of strings

- Only printing those shorter than 3 (in this test case).

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

Swift

```swift
import Foundation

class Erase
{
  func main()
  {
    var names:String[] = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names:String[] = filterLongerThan(names, length:3)
    for name:String in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result:String[] = String[]()
    for s:String in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase:Erase = Erase()
erase.main()
```

Swift

- Type Inference

```swift
import Foundation

class Erase
{
  func main()
  {
    var names = String[]()
    names.append ("ted")
    names.append ("fred")
    names.append ("jed")
    names.append ("ned")
    println(names)
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Literals

```
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = filterLongerThan(names, length:3)
    for name in short_names
    {
      println (name)
    }
  }

  func filterLongerThan (strings : String[], length : Int) -> String[]
  {
    var result = String[]()
    for s in strings
    {
      if countElements(s) < length + 1
      {
        result.append(s)
      }
    }
    return result
  }
}

var erase = Erase()
erase.main()
```

Swift

- Closures

```swift
import Foundation

class Erase
{
  func main()
  {
    var names = ["ted", "fred", "jed", "ned"]
    var short_names = names.filter { countElements($0) < 4 }
    for name in short_names
    {
      println (name)
    }
  }
}

var erase = Erase()
erase.main()
```

Swift

- Final version

```
import Foundation

var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

```java
import java.util.ArrayList;
import java.util.List;

class Erase
{
  public static void main(String[] args)
  {
    List<String> names = new ArrayList<String>();
    names.add("Ted");
    names.add("Fred");
    names.add("Jed");
    names.add("Ned");
    System.out.println(names);
    Erase e = new Erase();
    List<String> short_names = e.filterLongerThan(names, 3);
    System.out.println(short_names.size());
    for (String s : short_names)
    {
      System.out.println(s);
    }
  }

  public List<String> filterLongerThan(List<String> strings, int length)
  {
    List<String> result = new ArrayList<String>();
    for (String s : strings)
    {
      if (s.length() < length + 1)
      {
        result.add(s);
      }
    }
    return result;
  }
}
```

```
names = ["Ted", "Fred", "Jed", "Ned"]
println names
short_names = names.findAll{it.size() <= 3}
short_names.each {println it}
```
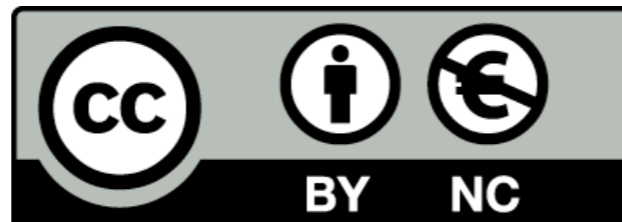
```
var names = #["Ted", "Fred", "Jed", "Ned"]
println(names)
var short_names =  names.filter[it.length() <= 3]
short_names.forEach[println(it)]
```

```
var names = ["ted", "fred", "jed", "ned"]
println(names)
var short_names = names.filter { countElements($0) < 4 }
println(short_names)
```

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit