ICT Skills Summer School

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

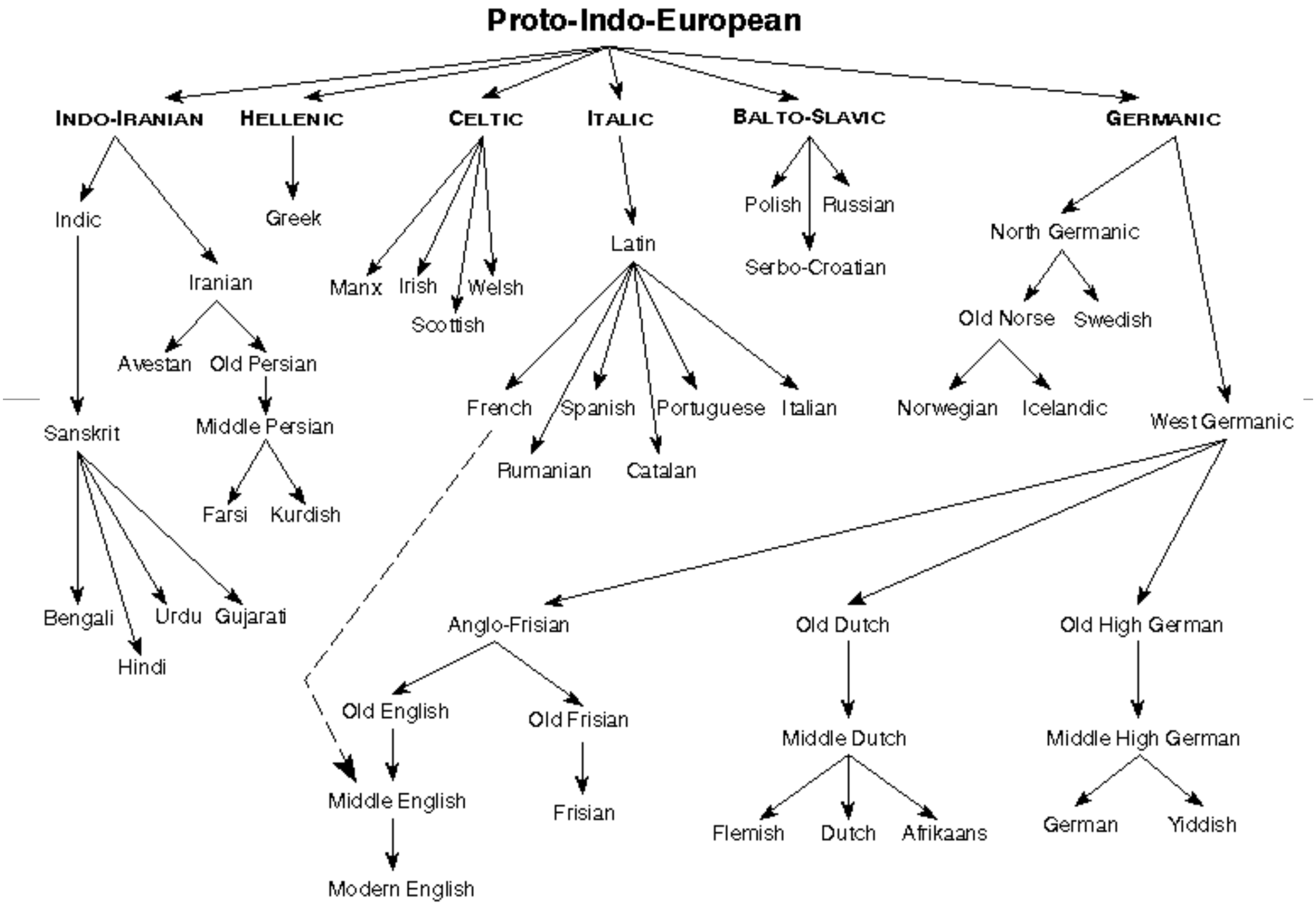
Waterford Institute *of* Technology

INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

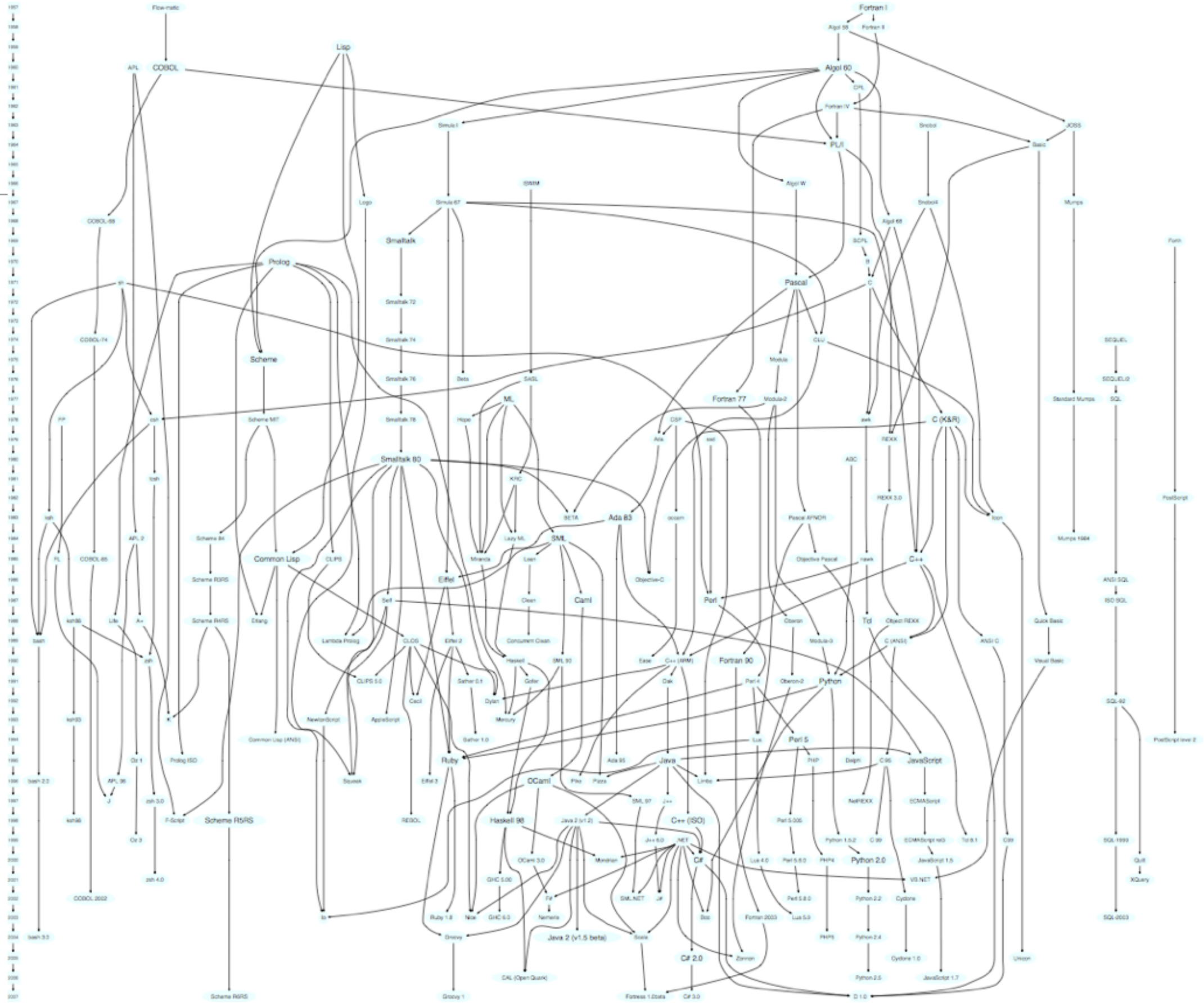
Typing in Programming Languages

Language Family Trees

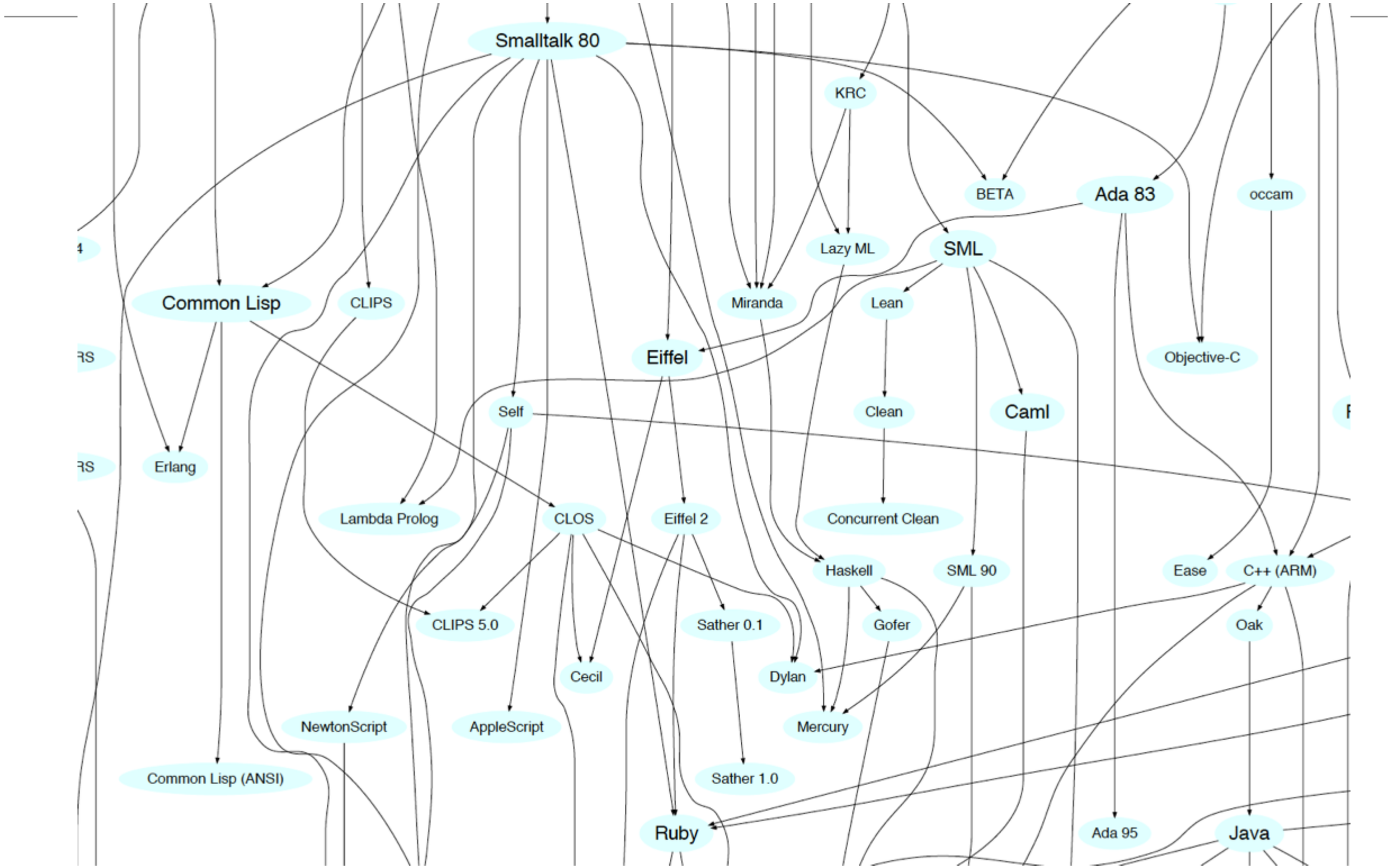


Prepared by Jack Lynch, jlynch@andromeda.rutgers.edu

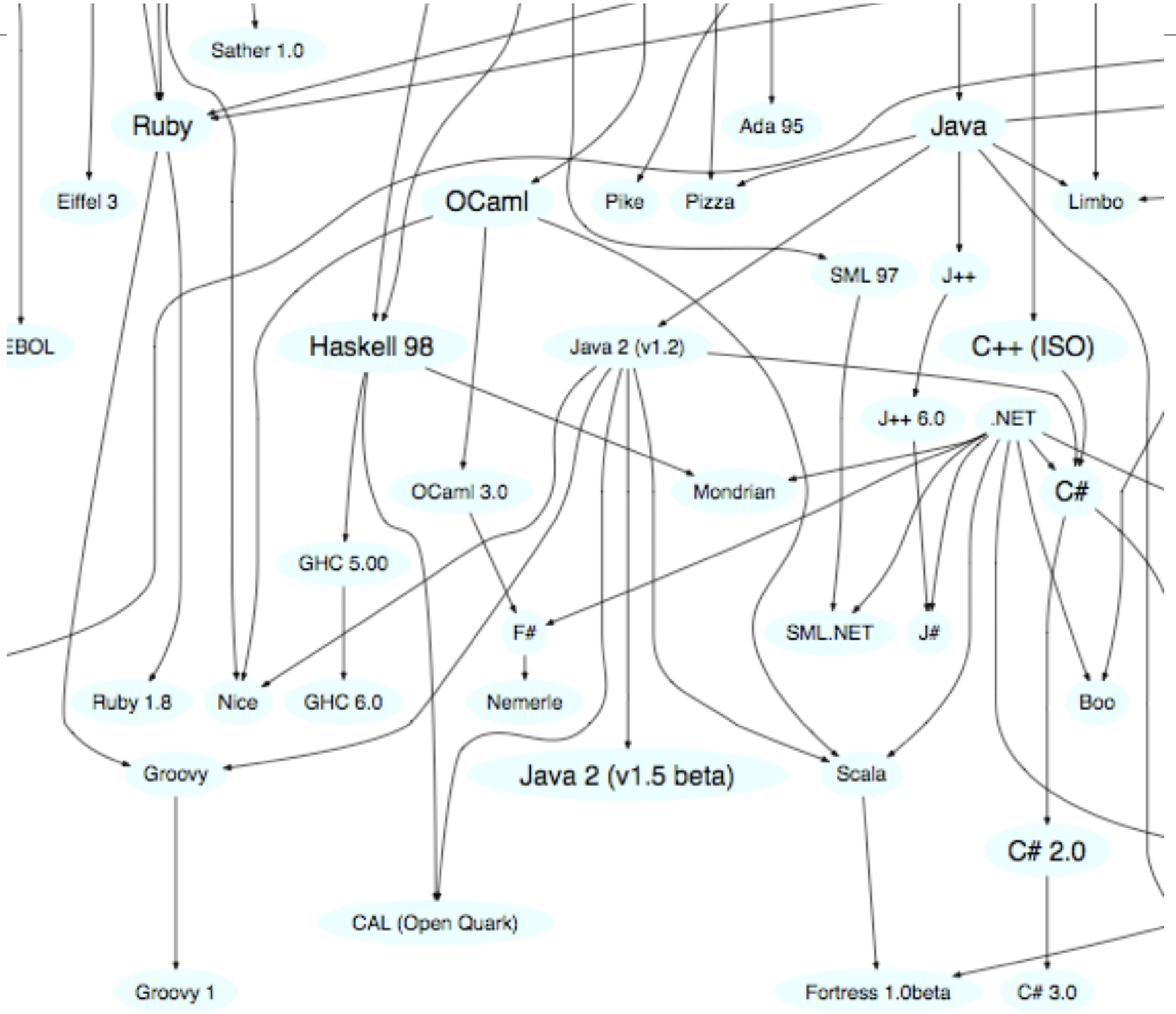
Family Tree (3)

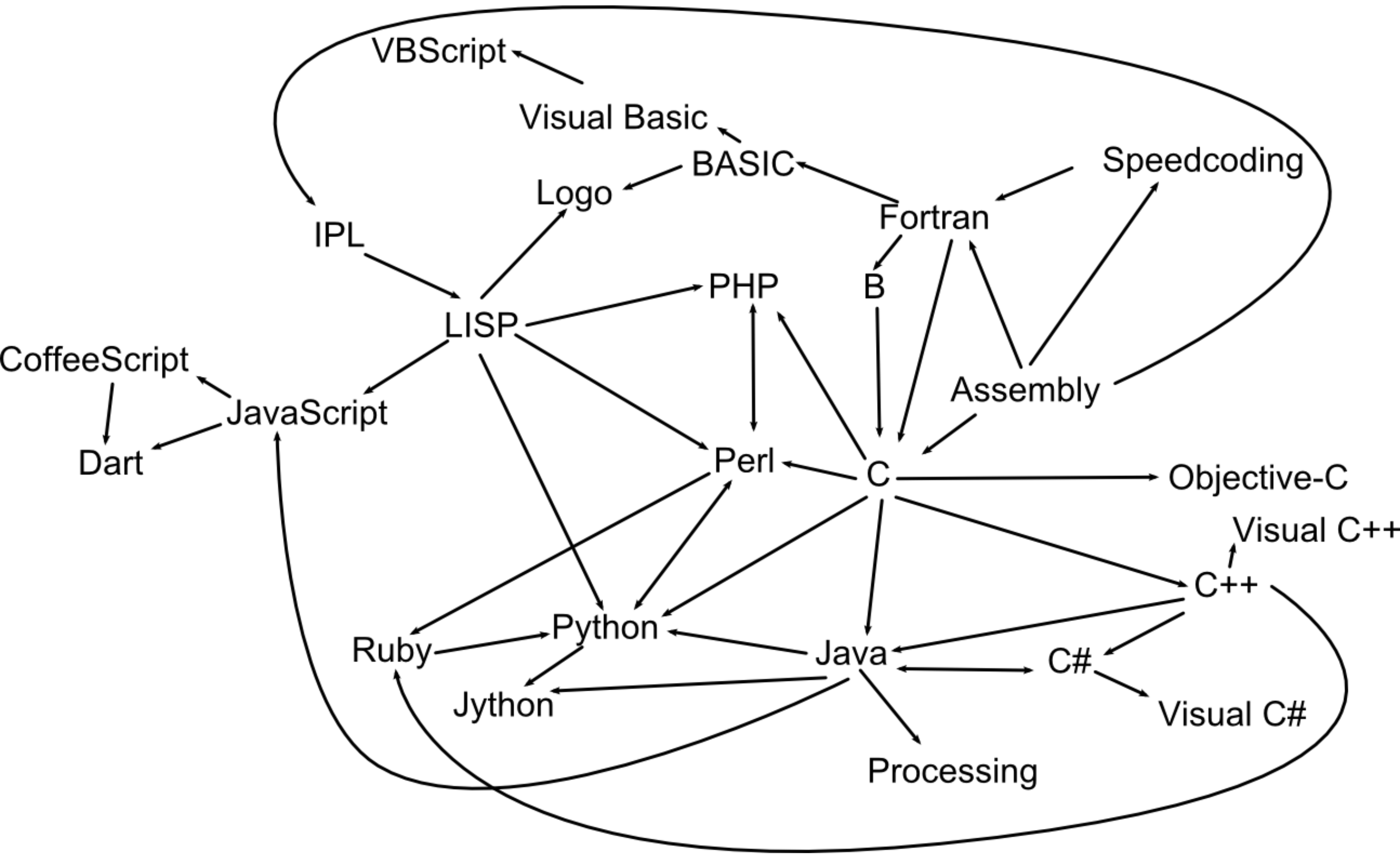


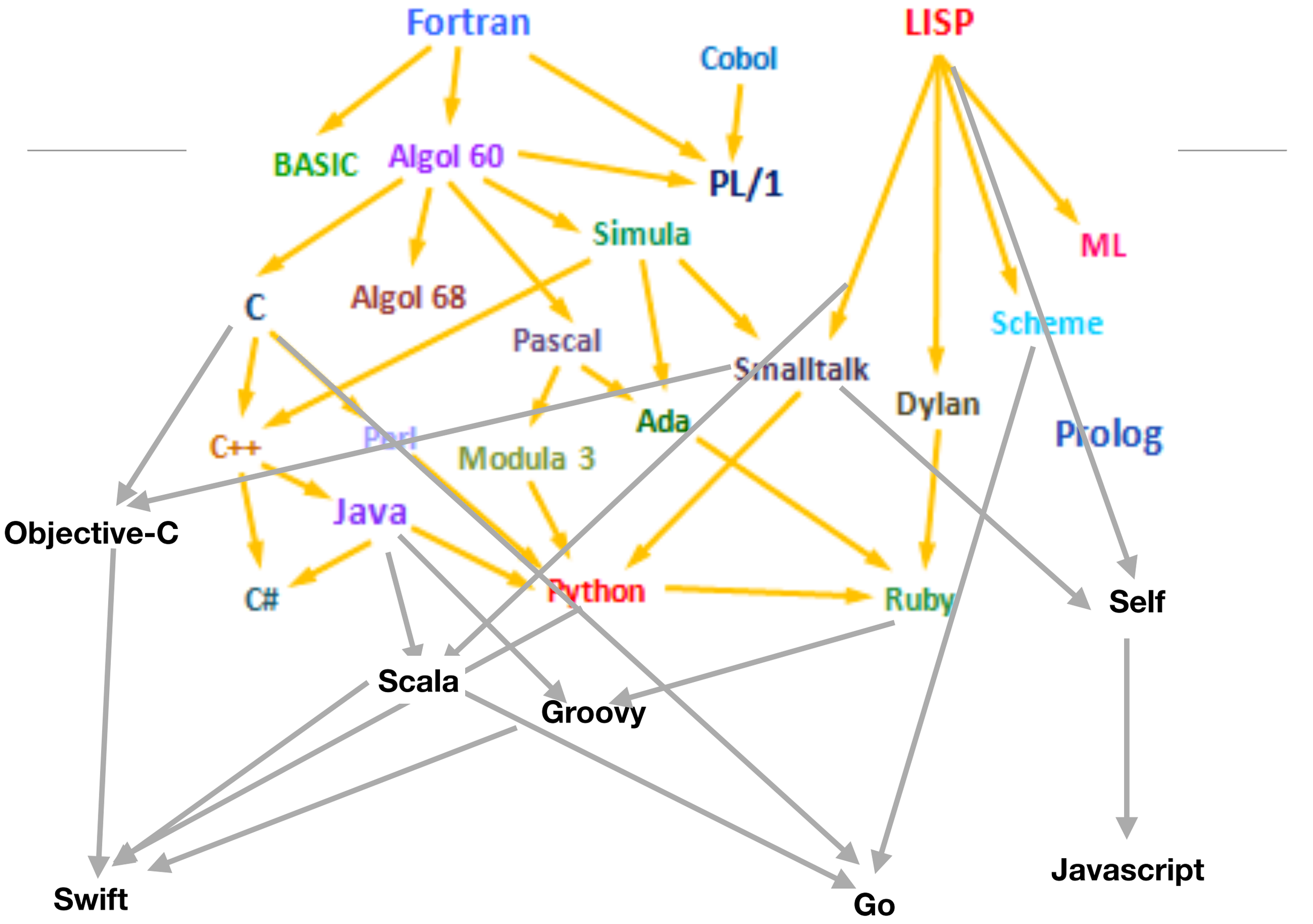
Smalltalk Cluster



Ruby, Groovy, Java, Scala Cluster




VBScript
Visual Basic
BASIC
Speedcoding
Logo
Fortran
IPL
PHP
B
LISP
CoffeeScript
Assembly
JavaScript
Dart
Perl
C
Objective-C
Visual C++
C++
Python
Ruby
Java
C#
Jython
Visual C#
Processing


Fortran
LISP
Cobol
BASIC
Algol 60
PL/1
Simula
ML
C
Algol 68
Scheme
Pascal
Smalltalk
Dylan
Ada
C++
Perl
Modula 3
Prolog
Objective-C
Java
C#
Python
Ruby
Self
Scala
Groovy
Javascript
Swift
Go

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 (Java 8 only)
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/Javascript

(from Neal Ford)

---

1. Conditionals
2. A function type
3. Recursion
4. Dynamic typing (+ 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/Javascript

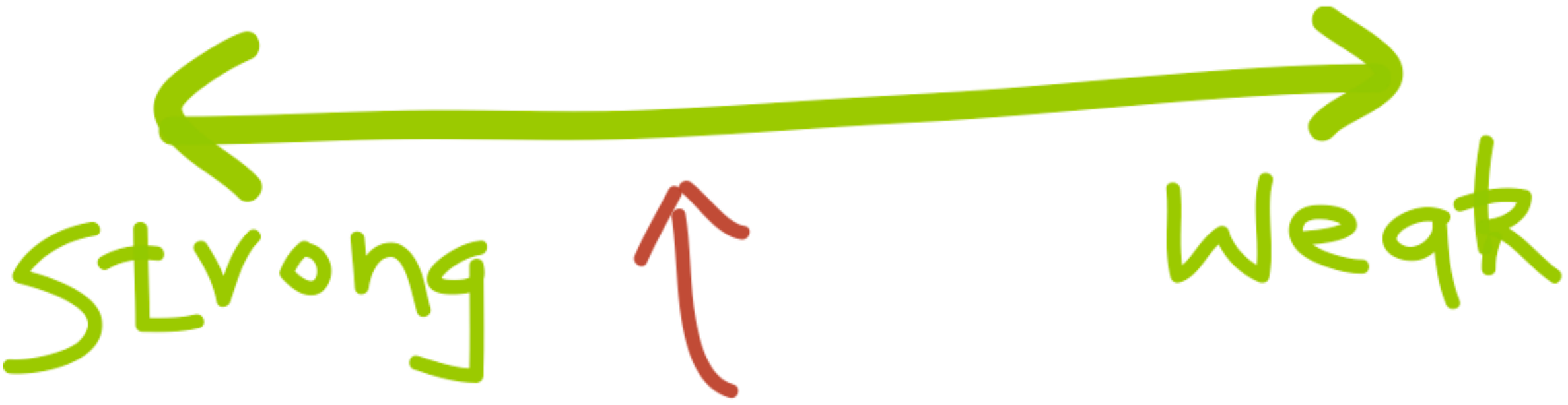
(from Neal Ford)

1. Conditionals
2. A function type
3. Recursion
4. Dynamic typing (+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

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



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

Another Approach to Types?

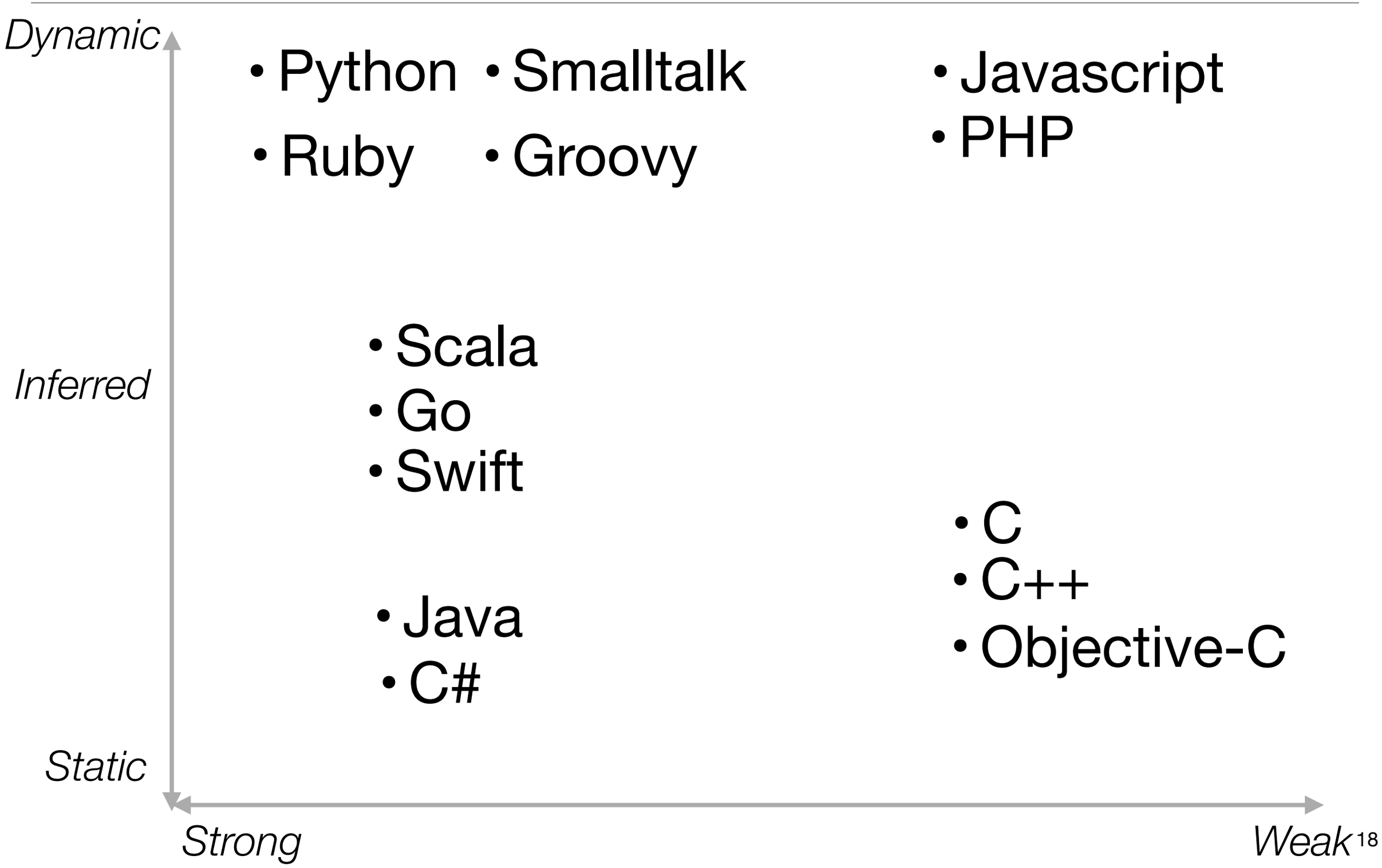
- *Type Inference* : the compiler draws conclusions about the types of variables based on how programmers use those variables.
    - Yields programs that have some of the conciseness of Dynamically Typed Languages
    - But - decision made at *compile time*, not at *run time*
    - More information for static analysis - refactoring tools, complexity analysis. bug checking etc...
- Haskell, Scala, Swift

```
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 va
}
```

‘Pragmatic’ Languages

- Python
- Ruby
- Smalltalk
- Groovy
- Javascript
- PHP
- Scala
- Go
- Swift
- Java
- C#
- C
- C++
- Objective-C

Typing Spectrum



Java Example

(from Jim Weirich)

- Java algorithm to filter a list of strings
- Only printing those shorter than 3 (in this test case).

```
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...

```
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?

```
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

```
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

```
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

```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
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?

Java Example (again)

```
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

```
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

```
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

```
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;
  }
}
```

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

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

Java Example (again)

```
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;
  }
}
```

Javascript

```
'use strict';

class Erase {

  static main ()
  {
    const names = [];
    names.push('Ted');
    names.push('Fred');
    names.push('Jed');
    names.push('Ned');
    console.log(names);
    const e = new Erase();
    const short_names = e.filterLongerThan(names, 3);
    console.log(short_names.length);
    for (const s of short_names)
    {
      console.log(s);
    }
  }

  filterLongerThan(strings, length)
  {
    const result = [];
    for (var s of strings)
    {
      if (s.length < length + 1)
      {
        result.push(s);
      }
    }
    return result;
  }
};

Erase.main();
```

Javascript

- Array Literals

```
'use strict';

const names = ['Ted', 'Fred', 'Jed', 'Ned'];
console.log(names);
const short_names = filterLongerThan(names, 3);
console.log(short_names.length);
console.log (short_names);

function filterLongerThan(strings, length)
{
  const result = [];
  for (var s of strings) {
    if (s.length < length + 1) {
      result.push(s);
    }
  }
  return result;
}
```

Javascript

- Lambdas

```
'use strict';

const names = ['Ted', 'Fred', 'Jed', 'Ned'];
console.log(names);
const short_names = filterLongerThan(names, 3);
console.log(short_names.length);
console.log (short_names);

function filterLongerThan(strings, length)
{
  let result = [];
  result = strings.filter (function (s)
  {
    return s.length < length + 1;
  });
  return result;
}
```

Javascript

- Arrow Functions


```
'use strict';

const names = ['Ted', 'Fred', 'Jed', 'Ned'];
console.log(names);
const short_names = filterLongerThan(names, 3);
console.log(short_names.length);
console.log (short_names);

function filterLongerThan(strings, length)
{
  let result = [];
  result = strings.filter (s => {
    return s.length < length + 1;
  });
  return result;
}
```

Javascript

- Final Version

```
'use strict';

const names = ['Ted', 'Fred', 'Jed', 'Ned'];
console.log(names);
const short_names = strings.filter (s => {
  return s.length < 4;
});
console.log(short_names.length);
console.log (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;
  }
}
```

Groovy

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

Swift

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

Javascript

```
'use strict';

const names = ['Ted', 'Fred', 'Jed', 'Ned'];
console.log(names);
const short_names = strings.filter (s => {
  return s.length < 4;
});
console.log(short_names.length);
console.log (short_names);
```

Another ‘Shopping List’

Object-literal syntax for arrays and hashes
Array slicing and other intelligent collection operators
Perl 5 compatible regular expression literals
Destructuring bind (e.g. x, y = returnTwoValues())
Function literals and first-class, non-broken closures
Standard OOP with classes, instances, interfaces, polymorphism, etc.
Visibility quantifiers (public/private/protected)
Iterators and generators
List comprehensions
Namespaces and packages
Cross-platform GUI
Operator overloading
Keyword and rest parameters
First-class parser and AST support
Type expressions and statically checkable semantics
Solid string and collection libraries
Strings and streams act like collections

http://steve-yegge.blogspot.com.au/2007/02/next-big-language.html

Java

Object-literal syntax for arrays and hashes	
Array slicing and other intelligent collection operators	
Perl 5 compatible regular expression literals	
Destructuring bind (e.g. x, y = returnTwoValues())	
Function literals and first-class, non-broken closures	
Standard OOP with classes, instances, interfaces, polymorphism,	y
Visibility quantifiers (public/private/protected)	y
Iterators and generators	y
List comprehensions	
Namespaces and packages	y
Cross-platform GUI	y
Operator overloading	
Keyword and rest parameters	
First-class parser and AST support	
Type expressions and statically checkable semantics	y
Solid string and collection libraries	y
Strings and streams act like collections	y

Google GO

Object-literal syntax for arrays and hashes	y
Array slicing and other intelligent collection operators	y
Perl 5 compatible regular expression literals	
Destructuring bind (e.g. x, y = returnTwoValues())	y
Function literals and first-class, non-broken closures	y
Standard OOP with classes, instances, interfaces, polymorphism,	
Visibility quantifiers (public/private/protected)	y
Iterators and generators	
List comprehensions	
Namespaces and packages	y
Cross-platform GUI	
Operator overloading	
Keyword and rest parameters	y
First-class parser and AST support	y
Type expressions and statically checkable semantics	y
Solid string and collection libraries	y
Strings and streams act like collections	

Python

Object-literal syntax for arrays and hashes	y
Array slicing and other intelligent collection operators	y
Perl 5 compatible regular expression literals	y
Destructuring bind (e.g. x, y = returnTwoValues())	y
Function literals and first-class, non-broken closures	y
Standard OOP with classes, instances, interfaces, polymorphism,	y
Visibility quantifiers (public/private/protected)	
Iterators and generators	y
List comprehensions	y
Namespaces and packages	y
Cross-platform GUI	
Operator overloading	
Keyword and rest parameters	y
First-class parser and AST support	y
Type expressions and statically checkable semantics	
Solid string and collection libraries	y
Strings and streams act like collections	y

Javascript (ES6/7 only)

Object-literal syntax for arrays and hashes	y
Array slicing and other intelligent collection operators	y
Perl 5 compatible regular expression literals	y
Destructuring bind (e.g. x, y = returnTwoValues())	y
Function literals and first-class, non-broken closures	y
Standard OOP with classes, instances, interfaces, polymorphism,	y
Visibility quantifiers (public/private/protected)	?
Iterators and generators	y
List comprehensions	y
Namespaces and packages	y
Cross-platform GUI	y
Operator overloading	
Keyword and rest parameters	
First-class parser and AST support	
Type expressions and statically checkable semantics	y
Solid string and collection libraries	y
Strings and streams act like collections	y

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/