Integrative Programming

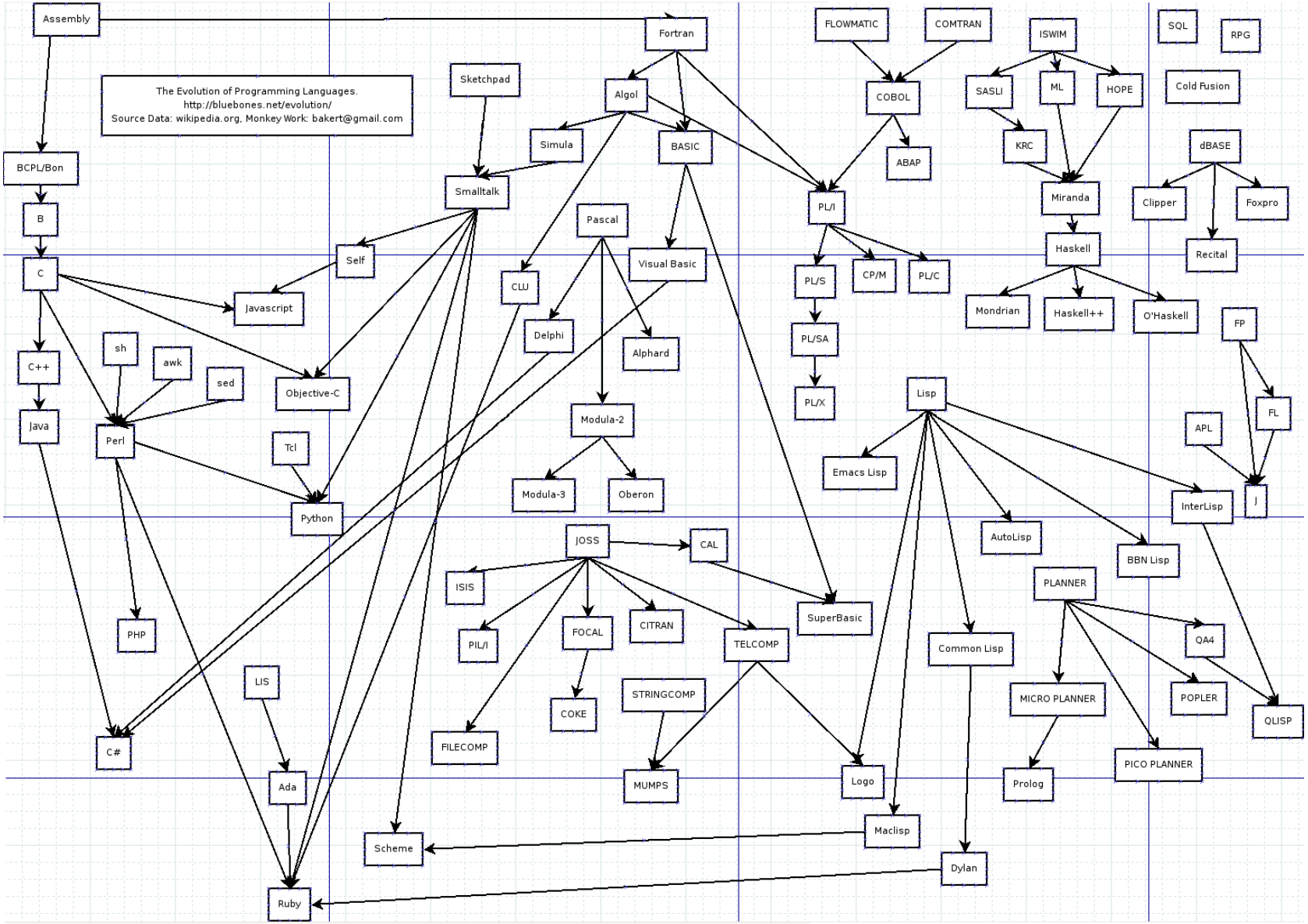
Definition

- Programming w/ purpose of combining and coordinating separate elements as to construct an interrelated whole
- Designing individual modules to function cooperatively as an entire system
- Incorporating modules coded in different languages to achieve unified task

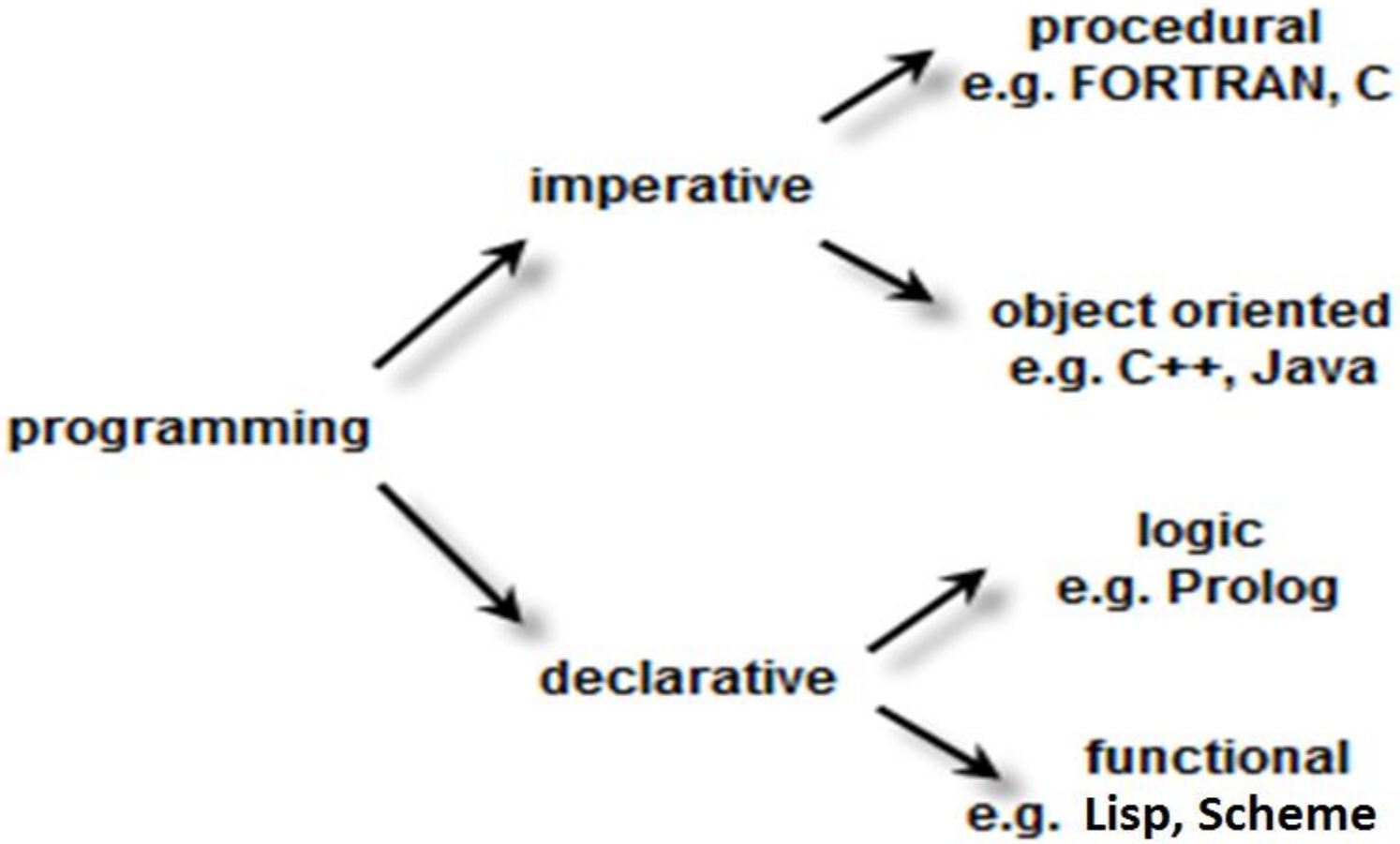
Programming Language

- An artificial language designed to communicate instructions to machine (computer)
- Used to create programs that control behavior of machine and/or express algorithms
- Defined by:
  - Syntax (combination of permitted symbols)
  - Semantics (arrangement of symbols such that they have meaning)

Evolution of Computer Languages



Traditional Programming Language Taxonomy



Compiled vs. Interpreted Language

- Compiled:
  - Code is reduced to set of machine instructions before being saved to executable file
  - Cannot be changed at runtime
  - Examples: C, C++, Java
- Interpreted:
  - Code is saved in same format as entered
  - Can change at runtime
  - Examples: Perl, Python, TCL, Ruby

Application vs. Scripting Language

- Application (System programming):
  - Programming language where programs are statically typed, allowing complex data structures, are compiled, and meant to operate largely independent of other programs
- Scripting:
  - Programming language where programs are dynamic in nature, and allow control of one or more applications and are interpreted from source code

Comparison of Programming Styles (C language)

```
#include <stdio.h>
int main(){
  printf(“Hello world!\n”);
  return 1;
}
```

- Semi-formal in nature
- Requires instantiating libraries –not all functions included when compiled/run (need “stdio.h” in this case to use printf)
- Requires compiling file then executing separate executable file
- Requires a main procedure (called “main”) to execute w/ return type
- Machine-dependent (different compilers on different machine have different specified requirements)

Comparison of Programming Styles (C++ language)

```
#include <iostream>
void main()
{
cout << "Hello World!";
}
```

Semi-formal in nature

- Requires instantiating libraries –not all functions included when compiled/run (need “iostream” in this case to use cout)
- Requires compiling file then executing separate executable file
- Requires a main procedure (called “main”) to execute w/ return type void
- Machine-dependent (different compilers on different machine have different specified requirements)

Comparison of Programming Styles (Java language)

```
class HelloWorld {
      public static void main(String args[]) {
        System.out.println("Hello world!");
      }
}
```

- Strictly formal in nature—Filename & Class name must match
- Requires declaring scope/nature of procedures
- Requires compiling file then executing separate executable file
- Requires a main procedure (called “main”) to execute w/o return type (void)
- Machine-independent (runs on a virtual machine environment—”same” for any Operating system)

Comparison of Programming Styles (Perl language)

```
print "Hello world!\n";
```

- Very informal in nature
- No file to compile—interpreted and executed immediately
- No procedures required to instantiate, no libraries required to call
- Driven toward appearance/presentation rather than structure

Comparison of Programming Styles (Python language)

```
print ("Hello world!")
```

- Very informal in nature
- No file to compile—interpreted and executed immediately
- No procedures required to instantiate, no libraries required to call
- Driven toward appearance/presentation rather than structure

Comparison of Programming Styles (VBscript)

```
<html>
 <body>
 <script type="text/vbscript">
 document.write("Hello world!")
 </script>
 </body>
 </html>
```

- Aside from tags, very informal in nature
- No procedures to instantiate, no libraries to call
- Driven toward appearance/presentation rather than structure

Comparison of Programming Styles (TCL)

- puts "Hello World!"

- No procedures to instantiate, no libraries to call
- Driven toward appearance/presentation rather than structure

Comparison of Programming Styles (Ruby)

In a text file:

```
puts "Hello World!“
```

At command prompt:

```
$ ruby -e "puts 'Hello world'"
```

- Can be very informal in nature; can be done “on-the-spot”
- Allows for Java-like format using objects
- No procedures to instantiate, no libraries to call
- Driven toward appearance/presentation rather than structure

Executable vs. Correct

- Executable:
  - Script file is able to be interpreted w/o any errors halting execution
- Correct:
  - Output is what is expected
- Errors:
  - Syntax/runtime errors: will be caught by interpreter
  - Logic errors: have to be corrected by user