17 October 2014

Programming Languages

Charlie London, Nikolai Smirnov, Jordan Spooner, Laurence Squires

Introduction

What is a Programming Language?

– Formal, constructed Language, which communicates instructions to a machine

Includes:

– Set of primitive instructions
– Set of primitive control structures
– Combining mechanisms

Used to:

– Create programs
– Control behavior of machine
– Express algorithms

Introduction

Comparing Programming Languages

- Syntax
- Static Semantics

- Specialized/ General Purpose
- Low-Level/ High-Level – Level of Abstraction
- Imperative/ Declarative
- Interpreted/ Compiled
- Strongly/ Loosely Typed
- Object – Oriented

Syntax

Which sequence of characters and symbols constitute a well-formed string

```
1  x = 3 + 4      ✓
2  x = 3 / 'Abc'  ✓
3  x = 3 4        ✘
```

Static Semantics

Which well-formed strings have a meaning

```
1  x = 3 / 1      ✓
2  x = 3 / 'Abc'  ✘
3  x = 3 * '5'    ?
```

Avoiding Ambiguity

e.g. `x = '5' + 6`

Different languages will assign a different value to 'x':

- One language might convert 6 to a string, and concatenate the two arguments to produce the string "56" (e.g. JavaScript)
- Another language might convert "5" to a number, and add the two arguments to produce the number 11 (e.g. Perl, PHP)
- Yet another language might convert the string "5" to a pointer representing where the string is stored within memory, and add 6 to that value to produce a semi-random address (e.g. C)
- And yet another language might simply fail to compile this program or run the code, saying that the two operands have incompatible type (e.g. Ruby, BASIC, Python)

DSL or GPL

Domain Specific (Specialized) Language

Suited to particular applications – e.g. HTML or Mathematica

General Purpose Language

Broadly applicable for use in multiple applications – e.g. C, Java, Python

Low-Level or High-Level

Low-Level Language

Provides little or no abstraction from machine code – Generally either machine code itself or assembly language

High-Level Language

Strong abstraction from machine code – Generally:

- Uses natural language elements
- Easier to use
- Automates/ hides completely significant areas of computing systems (e.g. memory management)

Imperative or Declarative

Imperative

Based on instructions, flow of control, and termination conditions which tell the computer **how** to perform a computation/ how to solve a problem (algorithms) – e.g. Python

Declarative

Describes what you want to do – not how to do it – i.e. specifying logic in terms of rules and facts e.g. SQL or Prolog

Interpreted or Compiled

Interpreted

Source Code → Checker → Interpreter → Output

(Better for debugging) – e.g. Python

Compiled

Source Code → Checker/ Compiler → Object Code → Interpreter → Output

(More efficient) – e.g. C

Strongly or Weakly Typed

Strongly Typed

Will require you to state type of every variable you define

(Generally less likely to have bugs)

Weakly Typed

Doe not require you to assign every variable a type.

(Often easier to work with)

Object-Oriented Programming Languages

- Concept of ‘objects,’ which have ‘data fields’ (attributes that describe the object) and ‘methods’ (define how the object should behave at runtime)
- ‘Objects’ are usually instances of classes; interact with each other in order to complete computation
- Examples include Java, C++, C#, Perl, Python, Ruby

C

Advantages:

- Good optimization
- Used in firmware programming
- Relatively easy to learn
- Lots of libraries

Disadvantages

- No OOPS
- No runtime checking
- No strict type checking
- No namespace
- No constructors / destructors

C++

Advantages:

- Everything C has and more
  - Stronger type checking
  - Operators instead of function calls
  - Extensible types (reuse code)
- Support for OOP

Disadvantages

- More complex
- No runtime checking
- No network, async I/O, graphics, concurrency, serialization

C#

Advantages

- Improvements to C++:
  - Native garbage collection
  - Treat class-methods as free functions (ignores *this* pointer argument)
  - No global functions or variables
  - Less error prone
- Compiled to an intermediate language (e.g. CIL)
- Cleaner and faster
- Portable

Disadvantages

- Largely for Microsoft Windows Environement
- Requires .NET framework
- Removed “unsafe” code functionalities (pointers)

Java

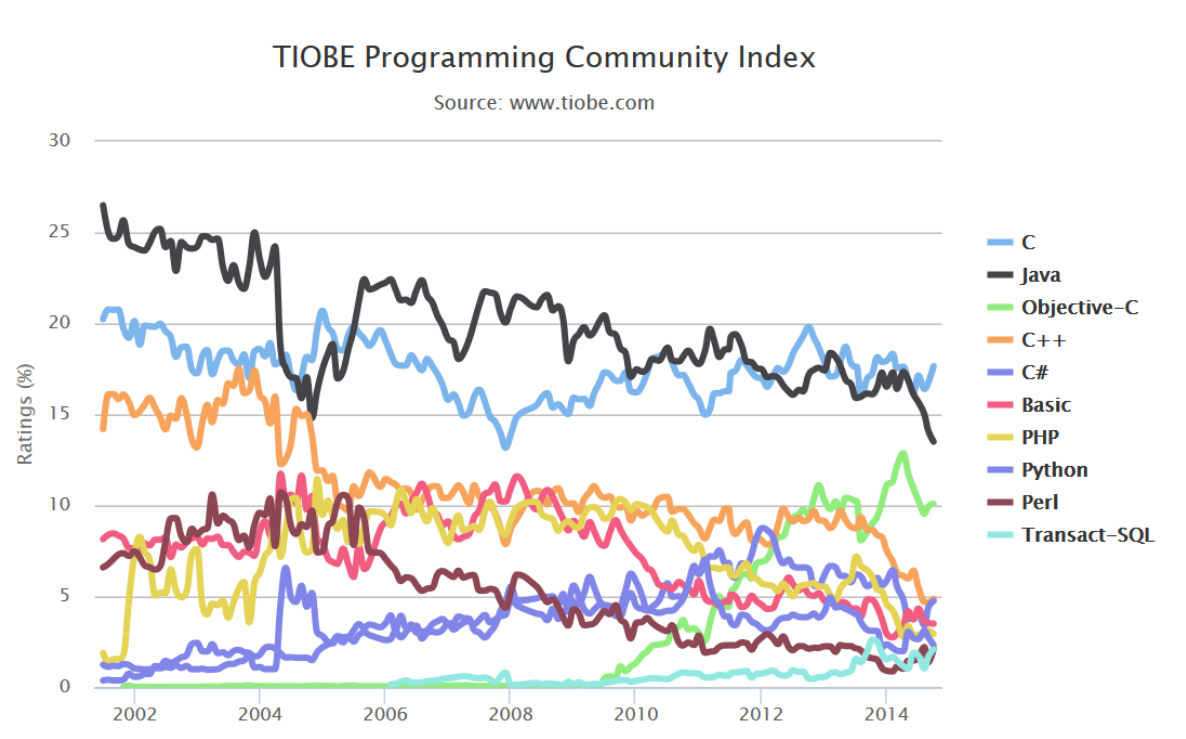
Overview

– Source code is a .java file, compiled into a .jar or .class and run by the JVM

– JVM is cross platform (over 3 billion devices…)

– Very popular language

– Is object orientated and has automatic memory handling



Java

Syntax

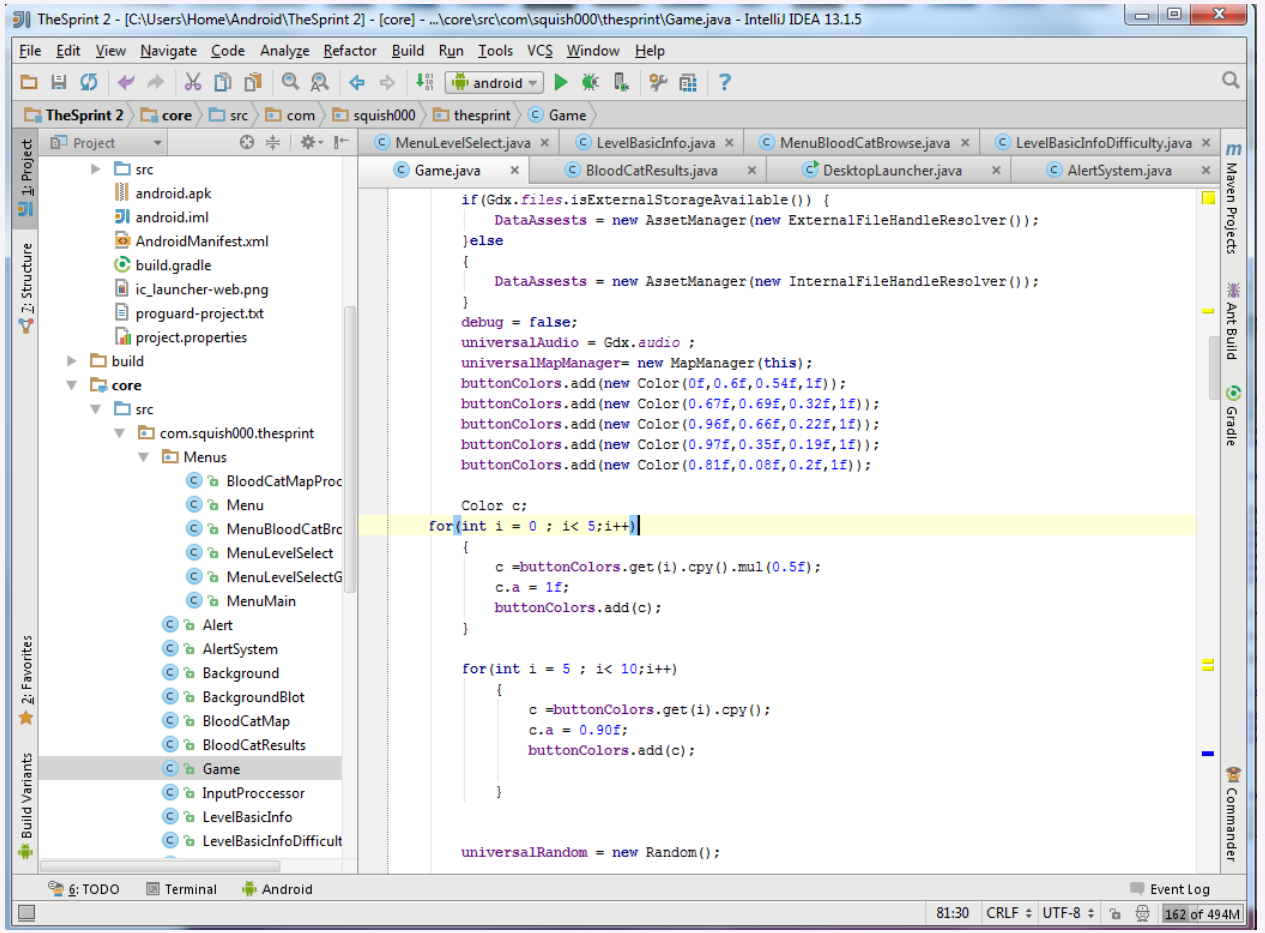
- Easy to read and understand
- IDE's have built in indenting and formatting functions

```
 public ArrayList<Color> buttonColors = new ArrayList<Color>();
public void addColours()
{
buttonColors.add(new Color(0.81f,0.08f,0.2f,1f));
    Color c;
    for(int i = 0 ; i< 5;i++)
        {
            c =buttonColors.get(i).cpy().mul(0.5f);
            c.a = 1f;
            buttonColors.add(c);
        }
}
```

Java IDE

Integrated Development Environment

- Java has good IDE and plugins support
- They allow for faster, more efficient coding
- Examples: Eclipse, IntelliJ, Netbeans…



Advantages

Libraries and Support

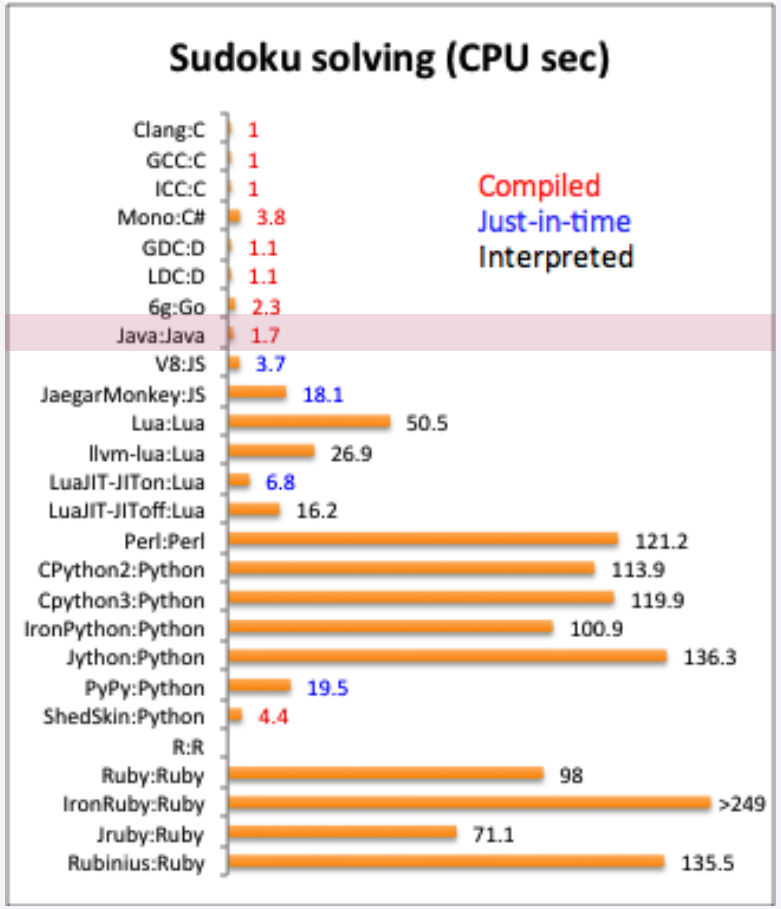
- Java has the largest amount of libraries and tutorials available, both 3rd party and oracle produced.

Cross Platform

- Runs on any OS with a JVM.

Speed

- Compiled languages are always fast
- Good error handling.



Disadvantages

Performance

- Compiling can be slow
- Applications have memory leaks
- Poor integration with the GPU

Complexity

- Large libraries cause bloated applications and security problems

HTML

Advantages:

- Plain text, very easy to edit
- Easy to pick up
- Widely used
- Fully supported by most web browsers

Disadvantages

- Static language
- Lack of security features
- Deprecated tags
- Limited styling capabilities without CSS (and CSS is a pain)
- IE being a pain as always

PHP

Advantages:

- Open Source; abundant extensions; stable
- Speed
- Easy to learn (mixed OOP and procedural paradigms; syntax similar to C)
- Widely supported
- Built in database connection modules

Disadvantages

- Bit of a mess: unpredictable, inconsistent, bloated, unreliable
- **Way** too many functions
- Not truly object oriented; stupid syntax
- Weakly typed – can lead to bugs

PHP

```
array_sum(array_map(function($object) {
    $object->foo();
}, $objects));
```

Ruby

```
objects.collect(&:foo).sum
```

JavaScript

Advantages:

- Executed client-side; quite fast
- Quite easy to learn and use
- Adds a lot of functionality to web pages

Disadvantages

- Execute client-side: security issues – users may choose to block as a result
- Differing layout engines – rendering varies leading to inconsistency

Lua

Overview

- Interpreted language
- Built around ANSI C (old, cross platform version of C)
- Designed to have no bloat
- Can interact with C languages, e.g. a C script can call a function from lua file

Lua

Syntax

- Easy to learn
- Annoying indenting

```
Point = {}

Point.new = function(x, y)
  return {x = x, y = y}  --  return {["x"] = x, ["y"] = y}
end

Point.set_x = function(point, x)
  point.x = x  --  point["x"] = x;
end
```

```
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
```

Advantages

Lightweight

- Produces small files, executes quickly
- Easy to create and executes on any OS

Non-bloated

- Developer chooses to add features.
- The framework is easily extendable.

C API

- Intended to be embedded in other applications
- Can easily change C variables and execute functions in C code

Disadvantages

Simple

- No provided class or object support
- Very little variety in data types

Weird

- Customizability is difficult to learn
- Limited error handling

Python

Example of Syntax

```
x = 12345
epsilon = 0.01
numGuesses = 0
low = 0.0
high = x
ans = (high + low)/2.0
while abs(ans**2 - x) >= epsilon and ans <= x:
    #print low, high, ans
    numGuesses += 1
    if ans**2 < x:
        low = ans
    else:
        high = ans
    ans = (high + low)/2.0
#print 'numGuesses =', numGuesses
print ans, 'is close to square root of', x
```

Python

Advantages

- One of the easiest languages to get started with
- Easy to write short programs
- ‘Syntax’ (or lack of) emphasizes code readability (fewer lines of code, forced indentation)
- Cross-Platform

Python

Disadvantages

– Sometimes too liberal

– Not extremely efficient (interpreted language)

Visual Basic

Pros/Uses

- Interactive and User Friendly - not case sensitive, Intellisense
- Simple - event driven, so there is no need to think or write sequentially
- Rapid Development - ready made controls, user friendly IDE
- Multiple Vendor Support - coordinates well with third party products
- Scripting Language - can be used with a server side programming language in web development
- VBA (Visual Basic for Microsoft Applications) - can create customised macros and program Microsoft Office products

Cons/Issues

- DLL (Dynamic Link Library) Issue - it can struggle using large numbers of DLLs, especially if some have conflicting names
- Memory Leakages - no proper automatic mechanism to handle untreated hanging objects which go out of scope
- Insufficient Web Development - limited performance in creating web applications
- Only for Windows - exclusive use
- Sluggish Performance - if network traffic and transaction volume is high it can slow down

Visual Basic

Example of Syntax

```
1  Mathematical Functions
2  Abs - Returns the absolute value of a number.
3  Acos - Returns the angle whose cosine is the specified number.
4  Asin - Returns the angle whose sine is the specified number.
5  Atan - Returns the angle whose tangent is the specified number.
6  Atan2 - Returns the angle whose tangent is the quotient of two specified numbers.
```

Haskell

Pros/Uses

- Strictly Typed - no null-type errors
- Lazy Evaluation - doesn't run code unnecessary for computation
- Functional - can write functions that manipulate other functions
- Parallel Programming Built In
- Multiplatform

Cons/Issues

- Programs take more work before they compile
- Obscure - few tutorials, minimal documentation
- Very Mathematical Community - lots of jargon, can be overwhelming
- More Difficult to Debug - lazy evaluation means you don't know whether a certain piece of code gets evaluated

Haskell

Example of Syntax

```
   Functions
1  (a >) partial application - (give the first argument to operator ">")
2  f a   partial application - (give the first argument)
3  (> a) partial application - (give the second argument to operator ">")
4  flip f b(2)    partial application - (give the second argument)
5  \a b -> ...    anonymous function
6  f a b ...      function call
7  f     function call (with no parameter)
8  .     function composition
9  f para1 para2 = ...     function definition
10 no syntax needed(3)    function return value (function body is the result)
11 id    identity function
```

Delphi

It's Shit

- The internet doesn't know
- No one knows
- No one cares