MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

# Programming Paradigms

# 159.272

# Programming Paradigms

## What are they?

Amjed Tahir
a.tahir@massey.ac.nz

# What is a Programming Paradigm ?

- a **programming paradigm** is a coherent set of principles used to design a **programming language**
- a programming paradigm is based on a certain **mathematical model of computation**
- different paradigms have strengths and weaknesses
- many programming languages implement principles from more than one paradigm

# Imperative Programming

- in **imperative programming**, a program consists of instructions that manipulate program state directly

  i.e., the programmer describes **how** the program is executed

- Examples: assembler, C, Java

# Imperative vs Declarative Programming

- In **declarative programming**, the programmer describes **what** he wants to achieve

- the actual  statements manipulating the computer program are **inferred** (by a compiler or an execution environment)

- Examples of declarative programming languages: Haskell, Prolog

# Imperative Programming

- imperative programming is based on the **Turing Machine** (or a similar) mathematical model

- **structured programming**: mandates the use of logical program structure (modules, conditionals, loops).

  Examples: C, Pascal - but not assembler

- **object-oriented programming (OOP)**: modern programming technique based on the idea of bundling data structure with functions manipulating these structures (objects)
- OOP facilitates **abstractions**

  OOP examples: Smalltalk, Ruby, Java, C++

# Declarative Programming

- **functional programming (FP)** is based on a different mathematical model, the lambda calculus

  computation is defined through the evaluation of functions, (global) state is avoided

  FP examples: Lisp, Haskell, Erlang, F#, Clojure

- **logic programming (LP)** is based on first-order predicate logic (FOL)

  a logic program consists of clauses (rules and facts) that are evaluated

  LP example: Prolog

# Current Trends (2018)

- OOP is still a mainstream

  it facilitates the engineering of large, modular applications

- OOP is particularly suitable for modelling (business) domains in a relatively intuitive way, and for programming user interfaces

- OOPs weakness is concurrency:

  One of the main weaknesses while concurrency is supported, it is tricky to synchronise access to shared state

# Current Trends (2017)

- FP is more suitable to write *scalable* and *concurrent* applications, and is therefore becoming more popular

this is driven by the trend towards multicore CPUs and distributed (cloud) computing

- trend towards "horses for courses" - mix and match different programming languages, or even use a **multi-paradigm programming language**

- Functional programming is growing!

# Moving towards *multiparadigm languages*

- many modern languages combine different paradigms - in particular OOP and FP

- classical example: JavaScript - a basically procedural language with functional and object-oriented features
- other examples: Scala, Ruby, Python

- ***Rust*** is a good example of a general purpose programming language.

# Example: the Evolution of Java

- Java started as a classical OOP language in the mid 90ties
- mid 2000: declarative features were added: annotations (e.g., annotate class as @persistent)
- single abstract method anonymous classes (SAM) are often used as functions (e.g. Google guava library)
- from 2013 - functional features are added to Java - functions are part of Java version 1.8

# Languages Covered in this Paper

- in this paper, we will cover two languages in great details:

  **Java - an imperative, object-oriented language**

  **&**

  **Scala - general-purpose programming language providing strong support for functional programming**

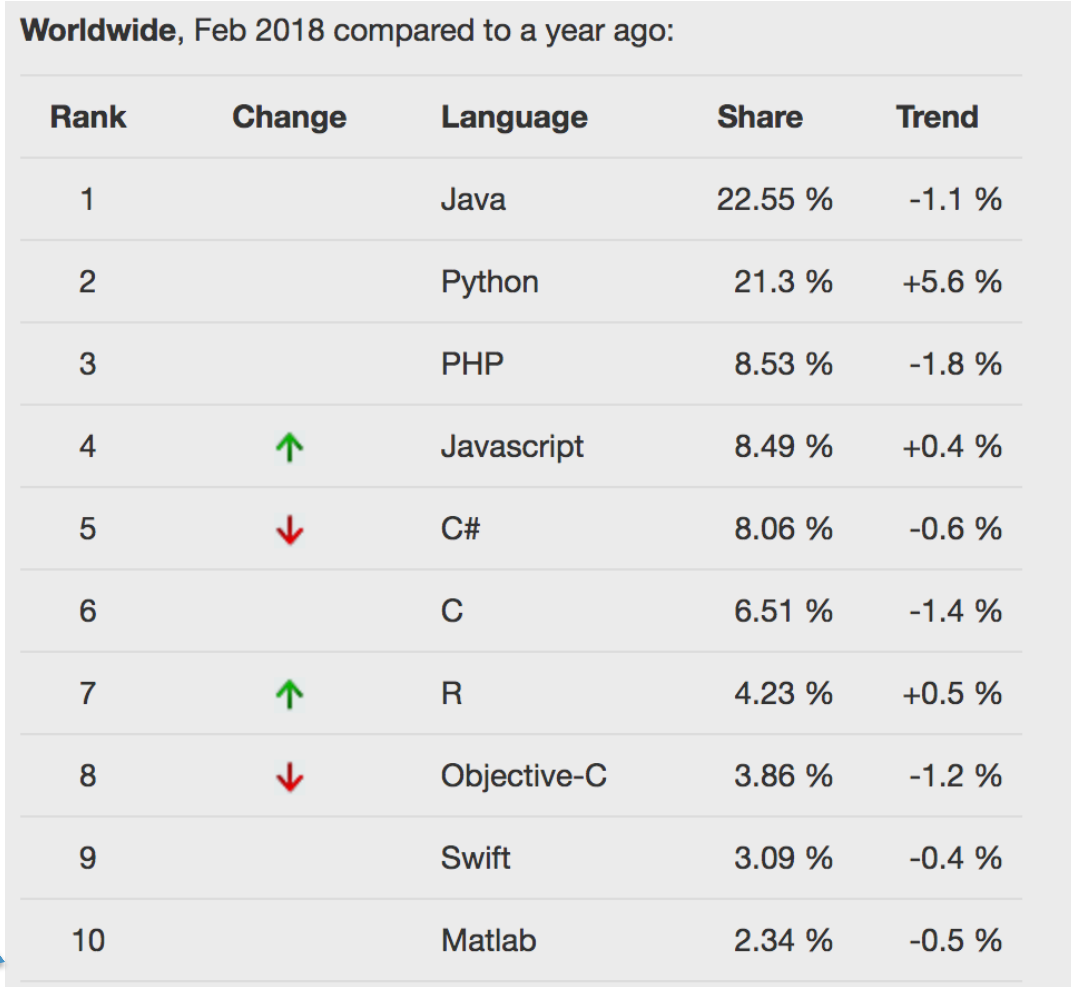
- there will be some references to other languages as well

# How popular are these languages?

## PYPL Popularity of Programming Language index

http://pypl.github.io/PYPL.html

Scala is number 15 in that list!

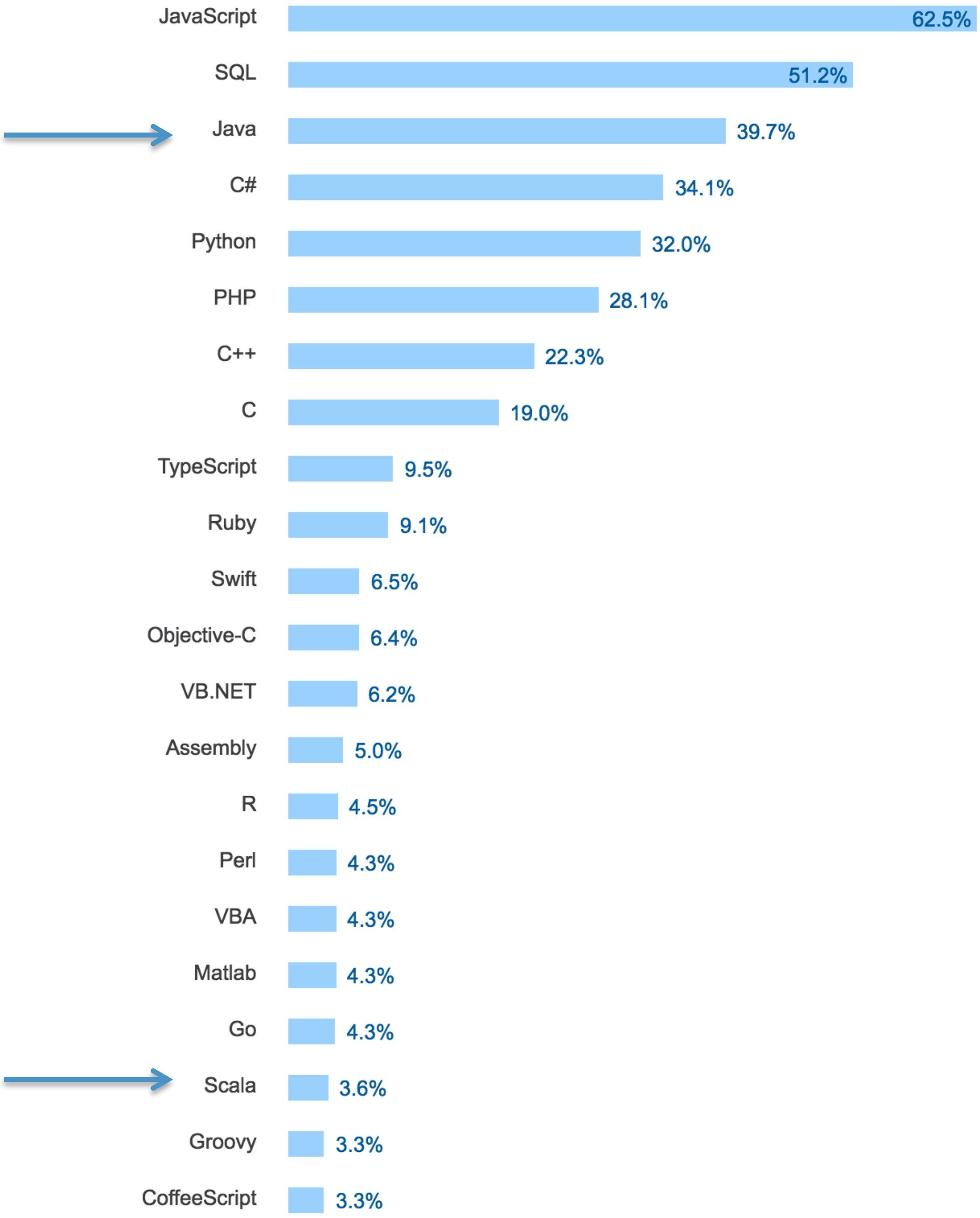


**Worldwide**, Feb 2018 compared to a year ago:

| Rank | Change | Language | Share | Trend |
|---|---|---|---|---|
| 1 | | Java | 22.55 % | -1.1 % |
| 2 | | Python | 21.3 % | +5.6 % |
| 3 | | PHP | 8.53 % | -1.8 % |
| 4 | ↑ | Javascript | 8.49 % | +0.4 % |
| 5 | ↓ | C# | 8.06 % | -0.6 % |
| 6 | | C | 6.51 % | -1.4 % |
| 7 | ↑ | R | 4.23 % | +0.5 % |
| 8 | ↓ | Objective-C | 3.86 % | -1.2 % |
| 9 | | Swift | 3.09 % | -0.4 % |
| 10 | | Matlab | 2.34 % | -0.5 % |

# Stack Overflow Developer Survey

[http://stackoverflow.com/research/developer-survey-2016#overview](http://stackoverflow.com/research/developer-survey-2016#overview)



# “Hello World!” in different languages

“Hello world” written in Python, C++ and Java

```
Print  (“hello
world”)
```

```
Void main ()
{
Printf ("\nHello world
\n");
}
```

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

# A brief history of programming languages

History of programming language goes with the history of computers.

It is hard to say which language was developed first.

Most sources refer to the period between 1945-1960 as where most ideas about programming came from.

# A brief history of programming languages

The first recorded language that was developed is Assembly (assembler) (very low-level!!)

Most of the early languages were high-level.

ShortCode (1949), AutoCode(1950), FORTRAN (1954) then LISP (1958) and COBOL (1959)

# Programming levels!

## High level vs low level

Can you name a few example of high level and low level programming languages?

# Programming levels!

High-level programming languages

Low-level programming languages

Intermediate (mixed)-level programming languages

# Low vs High level languages

- Low level programming languages

Close to the architecture level of compters.
Deal directly with hardware.
Provides very little (or even no!) abstraction. From computer’s instruction set architecture (i.e., the interface between software and hardware).

Example: Assembly

# Example from a low-level programming language.

- “Hello world!” in assembly

```
section     .text
   global _start     ;must be declared for linker (ld)

_start:                     ;tells linker entry point
   mov      edx,len     ;message length
   mov      ecx,msg     ;message to write
   mov      ebx,1       ;file descriptor (stdout)
   mov      eax,4       ;system call number (sys_write)
   int      0x80        ;call kernel

   mov      eax,1       ;system call number (sys_exit)
   int      0x80        ;call kernel

section     .data
msg db 'Hello world!', 0xa  ;string to be printed
```

# ***Low*** **vs** ***High*** **level programming**

- High-level programming language

  Comes with strong ***abstraction*** (hides the details of the computer – to reduce complexity)

  The higher the *abstraction*, the *higher the language*

  In many cases, it uses natural language
  Heavy use of automation, when possible.

○There is also high-level languages, and higher level languages (known as very high-level languages)

- High level: Java, C++ etc.…
- Very high level: most scripting languages such as Perl, Python, Ruby etc.…

# Example from a high-level programming language.

Ruby

```
puts 'Hello, world!'
```

Perl

```
Print 'Hello, world!’;
```

# Mixed level languages

Some languages are classified as mixed level languages

- 
  They are mainly high-level languages, with support of low=level programming
- Make them very useful, but increase the level of learning and dealing with these programs.

Examples: C++ and Rust (quite new!)

# Languages for different purposes?

Here is a nice infograph on the  history of programming languages, with some description

http://www.veracode.com/blog/2013/04/the-history-of-programming-languages-infographic

# Where to write your program?

IDEs, PyCharm?

you remember PyCharm from last year, right? That is what we call an “integrated development environment” (IDE).

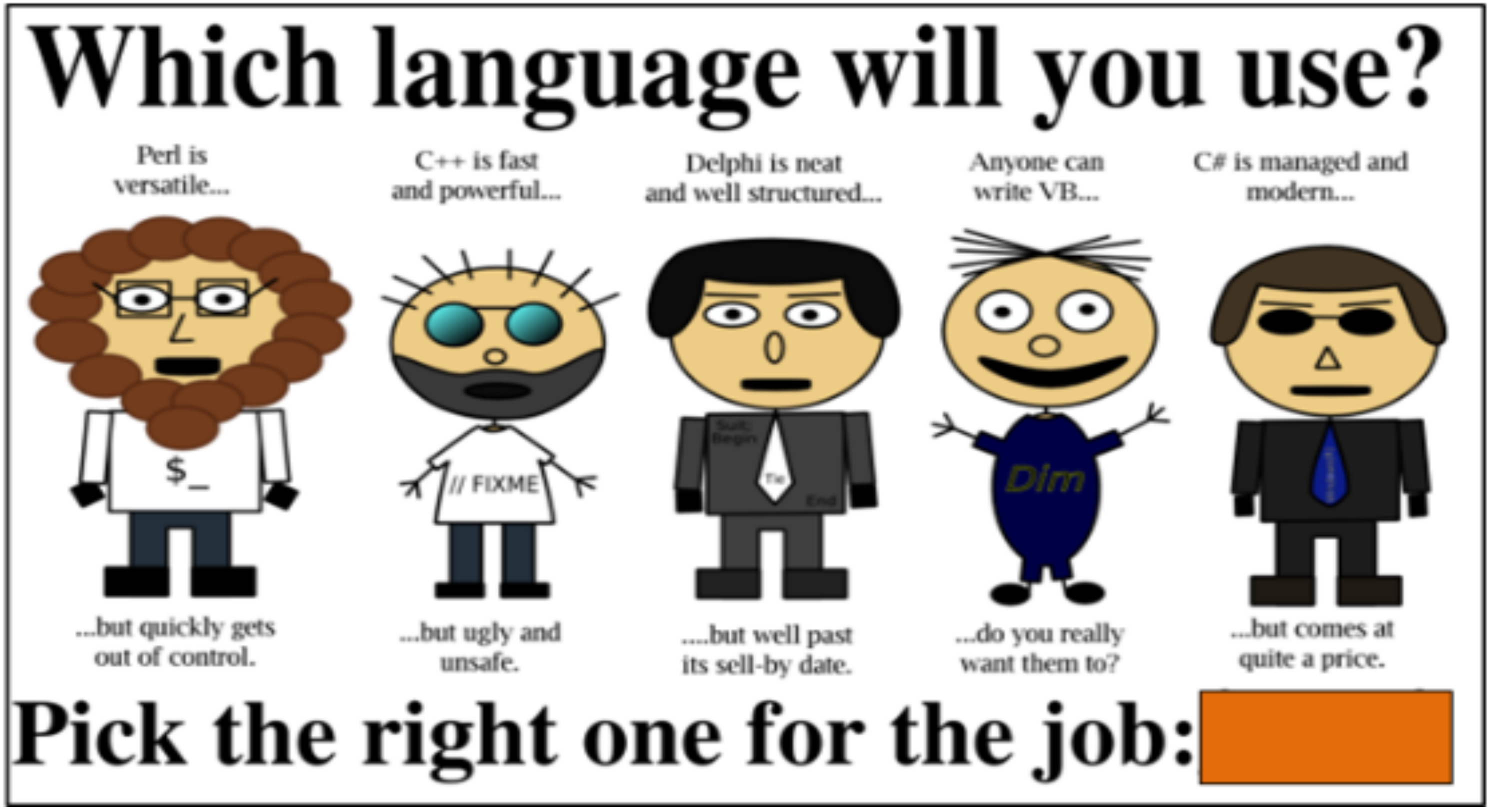
Almost all programs can be written using standard text editors

# Which language should you use?

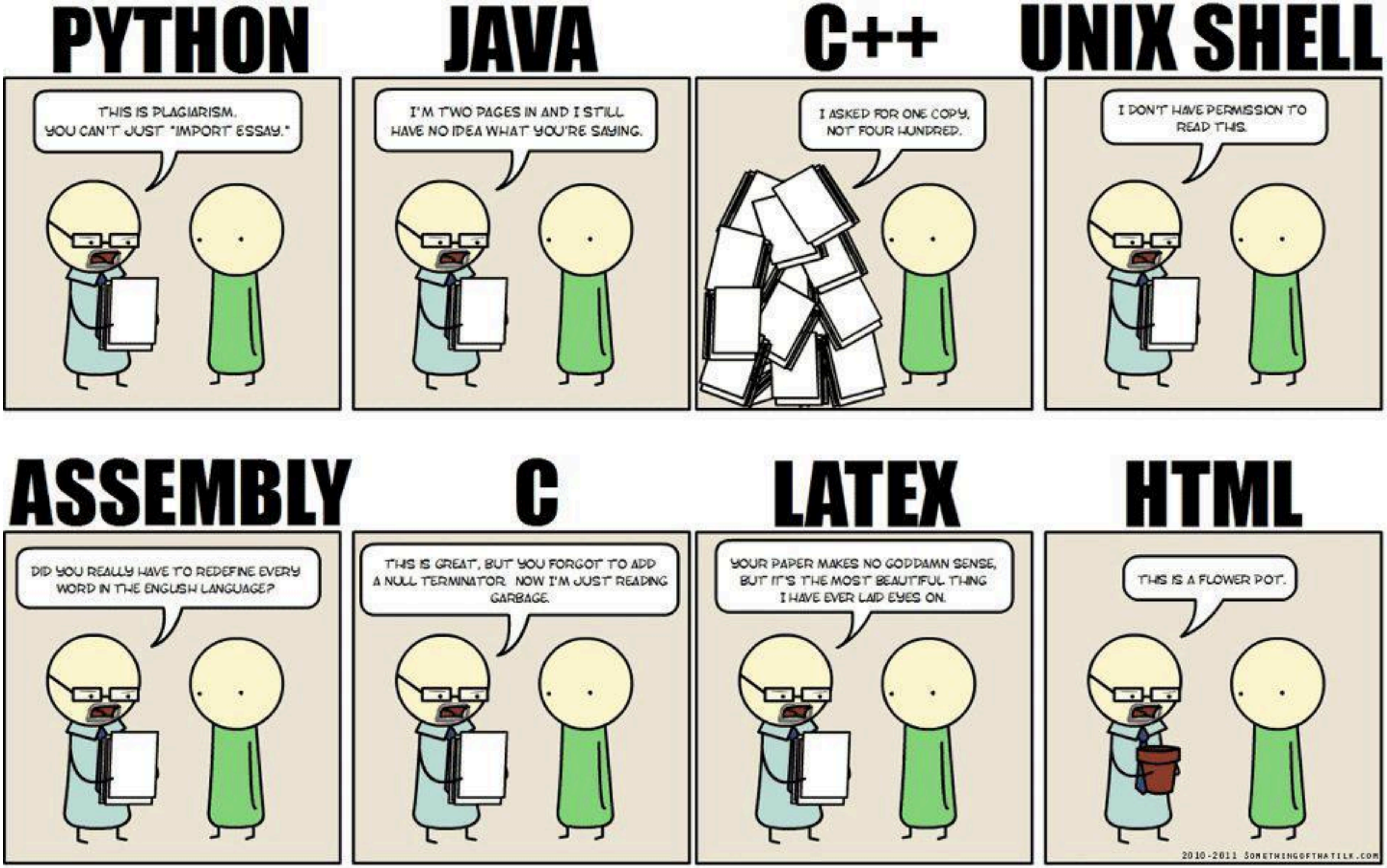
- The answer is simple: it depends on the purpose of the program you are writing, the time required to write and the resources available!!

- For example, if you are writing an iOS program, then Swift would be an excellent option.
- If you write something to work only on a Microsoft platform, then .NET languages would be a good option
- If you want something that works in a multicore processing enviournment, then you can a functional programming langugage like Haskell or Scala.

# Just for fun!!!


Which language will you use?
Perl is versatile...
C++ is fast and powerful...
Delphi is neat and well structured...
Anyone can write VB...
C# is managed and modern...
$_
// FIXME
Suit: Begin
Tie
End
Dim
...but quickly gets out of control.
...but ugly and unsafe.
....but well past its sell-by date.
...do you really want them to?
...but comes at quite a price.
Pick the right one for the job:

But could be true!


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM