Introducción a la Programación

Grado en Ingeniería Informática

Angel García Olaya
Planning and Learning Group (PLG)
Departamento de Informática
Escuela Politécnica Superior
Universidad Carlos III de Madrid
agolaya@inf.uc3m.es

2020/2021

Índice

- **¿Qué es programar?**
- Breve introducción a la arquitectura de un ordenador
- Lenguajes de Programación
- Paradigmas de Programación

¿Qué es programar?

- Una definición no muy formal:
  - Proporcionar a un ordenador un conjunto de **datos** y unas **instrucciones** sobre lo que se debe hacer con esos datos con el objetivo de resolver algún problema
- Las instrucciones explican cómo se debe operar con los datos para resolver el problema

¿Qué es programar? (conceptos importantes)

- Para crear un programa necesitamos:
  - Un **problema** (computable)
  - **Datos**, que caractericen/representen el problema
  - Un **algoritmo**, que detalle los pasos a seguir
  - Un **lenguaje de programación**, que usaremos para "explicar" el algoritmo al ordenador
- **Algoritmo:** lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema

Ejemplo de algoritmo

- Algoritmo para cambiar una rueda pinchada

  - PASO 1. Aflojar los tornillos de la rueda pinchada con la llave inglesa
  - PASO 2. Colocar el gato mecánico en su sitio
  - PASO 3. Levantar el gato hasta que la rueda pinchada pueda girar libremente
  - PASO 4. Quitar los tornillos
  - PASO 5. Quitar la rueda pinchada
  - PASO 6. Poner rueda de repuesto
  - PASO 7. Poner los tornillos y apretarlos ligeramente
  - PASO 8. Bajar el gato hasta que se pueda liberar
  - PASO 9. Sacar el gato de su sitio
  - PASO 10. Apretar los tornillos con la llave inglesa

Resolución de problemas de programación

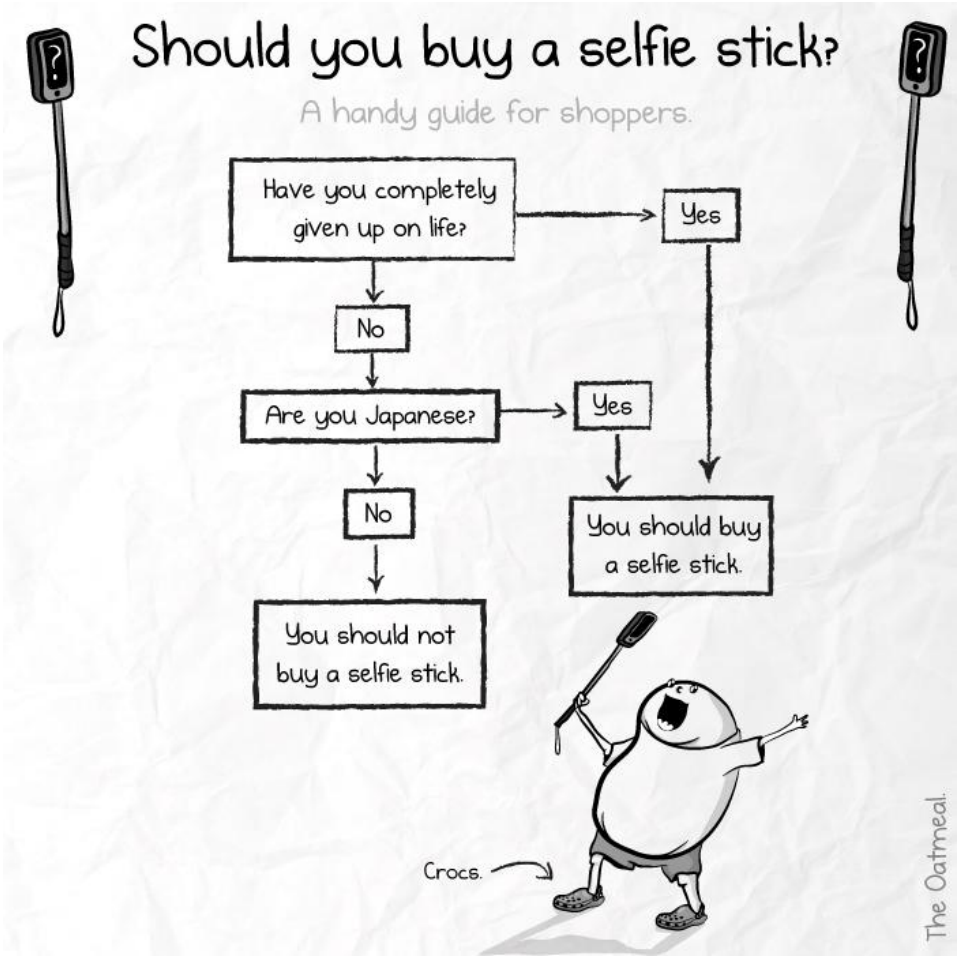
- El objetivo cuando se escribe un programa es resolver un problema
- Pasos generales para resolver problemas
  - Entender el problema
  - Si el problema es grande partirlo en piezas manejables
  - Diseñar una solución (un algoritmo)
  - Implementar (codificar) la solución
  - Probar la solución y reparar errores

Diseño de algoritmos

- Necesitamos expresar el algoritmo en un lenguaje que el ordenador entienda
  - El ordenador seguirá el algoritmo (ejecutará o correrá el programa) para resolver el problema
- ¡El primer paso es siempre diseñar el algoritmo!
  - Salvo que el algoritmo sea trivial no es recomendable codificarlo directamente
- Los programadores usan
  - Diagramas de flujo
  - Pseudo-código

Diagramas de flujo

- Representación gráfica de un algoritmo



Pseudo-código

- Descripción de un algoritmo en un lenguaje parecido a los lenguajes de programación reales de forma que luego resulte sencillo codificarlo
  - Mezcla de instrucciones genéricas y símbolos matemáticos

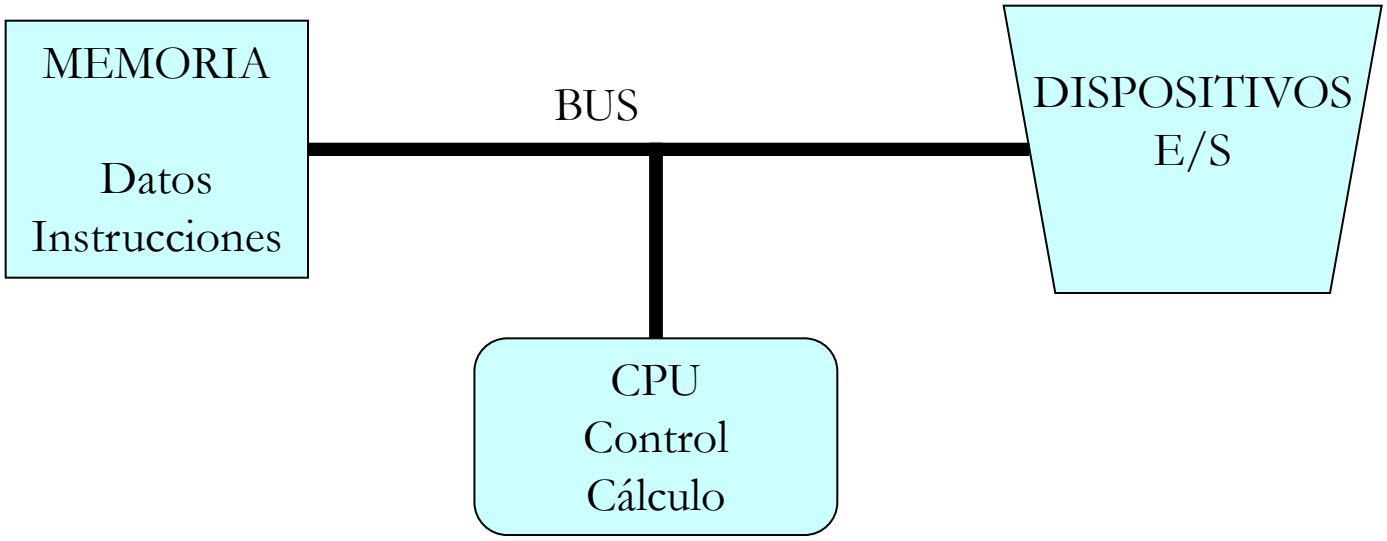
```
1: Plan-for-goals (P',N,H,u_f): Π plan
2: repeat
3:    Π ← plan(P')
4:    if Π ≠ ∅ then
5:       H ← G'
6:       return Π
7:    else
8:       N ← N ∪ G'
9:       G' ← remove one goal from G'
10:   end if
11: until G' = ∅ or utility(G') ≤ u_f
12: return ∅
```

Índice

- ¿Qué es programar?
- **Breve introducción a la arquitectura de un ordenador**
- Lenguajes de Programación
- Paradigmas de Programación

Arquitectura básica de un ordenador

- Hardware y Software
- La inmensa mayoría de los ordenadores (incluidos todos los personales) siguen esta arquitectura

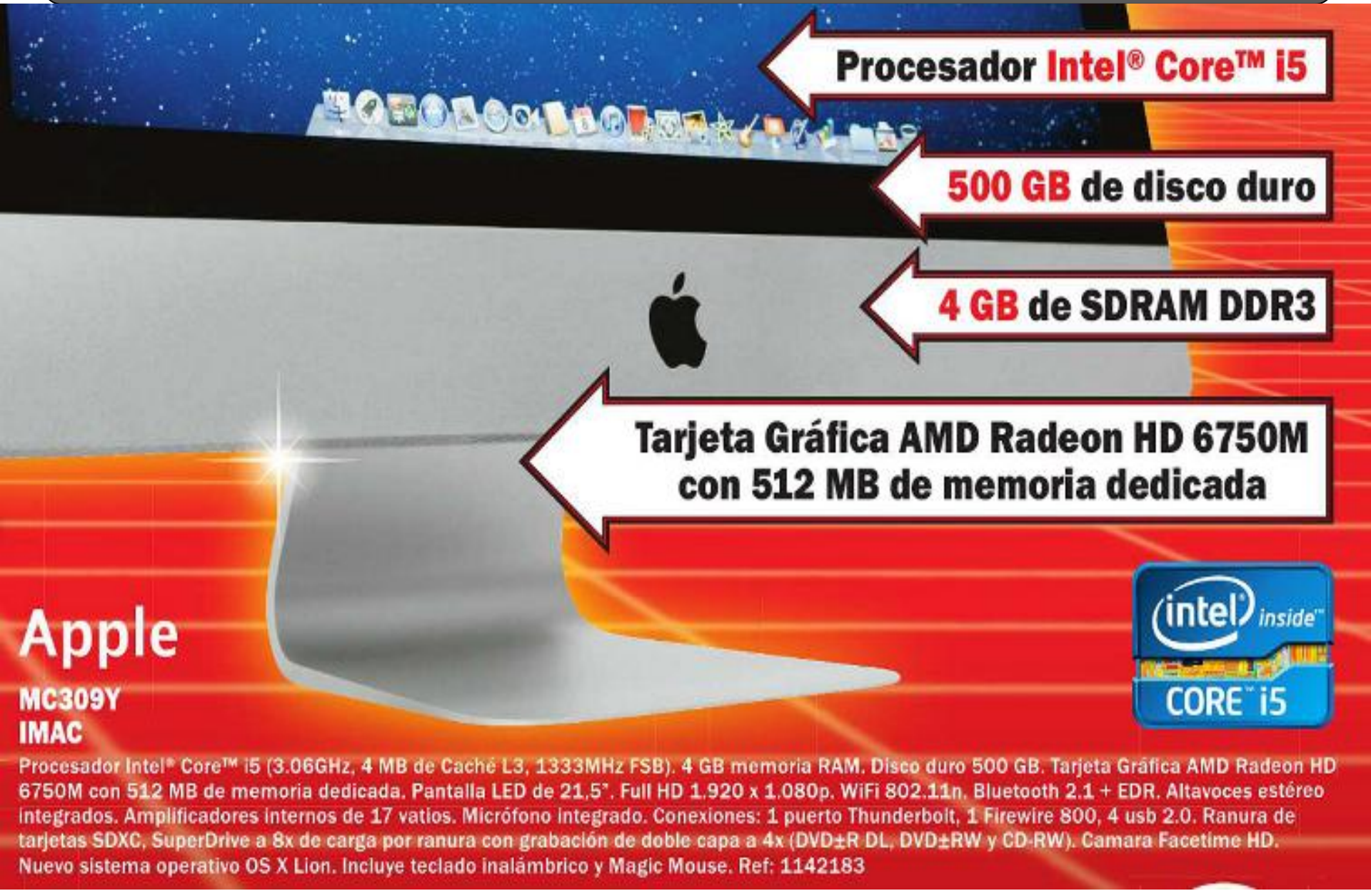


- Datos e instrucciones comparten memoria
- Propuesta inicialmente por Eckert y Mauchly aunque se conoce como arquitectura de Von Neumann

Componentes de la arquitectura

- Unidad Central de Procesamiento (CPU)
  - Ejecuta las instrucciones y coordina el resto de componentes
- Memoria
  - Guarda los datos, las instrucciones y los resultados
  - Principal/Secundaria
  - Permanente/Volátil
  - De acceso directo/secuencial
- Dispositivos de Entrada/Salida
  - Para comunicarse con el usuario o con otros sistemas
- Bus de datos
  - Para compartir la información entre los componentes anteriores

Ejemplo


Procesador Intel® Core™ i5
500 GB de disco duro
4 GB de SDRAM DDR3
Tarjeta Gráfica AMD Radeon HD 6750M
con 512 MB de memoria dedicada
intel inside
CORE i5
Apple
MC309Y
IMAC
Procesador Intel® Core™ i5 (3.06GHz, 4 MB de Caché L3, 1333MHz FSB). 4 GB memoria RAM. Disco duro 500 GB. Tarjeta Gráfica AMD Radeon HD 6750M con 512 MB de memoria dedicada. Pantalla LED de 21,5". Full HD 1.920 x 1.080p. WiFi 802.11n. Bluetooth 2.1 + EDR. Altavoces estéreo integrados. Amplificadores internos de 17 vatios. Micrófono integrado. Conexiones: 1 puerto Thunderbolt, 1 Firewire 800, 4 usb 2.0. Ranura de tarjetas SDXC. SuperDrive a 8x de carga por ranura con grabación de doble capa a 4x (DVD±R DL, DVD±RW y CD-RW). Camara Facetime HD. Nuevo sistema operativo OS X Lion. Incluye teclado inalámbrico y Magic Mouse. Ref: 1142183

Software

- Software de Sistema
  - Proporciona control sobre el hardware y sirve de base a las aplicaciones
- Software de Aplicaciones
  - Programas con finalidades específicas, resuelven un problema o familia de problemas determinados
  - Ofimática
  - Contabilidad
  - Diseño
  - Juegos
  - IA…

Índice

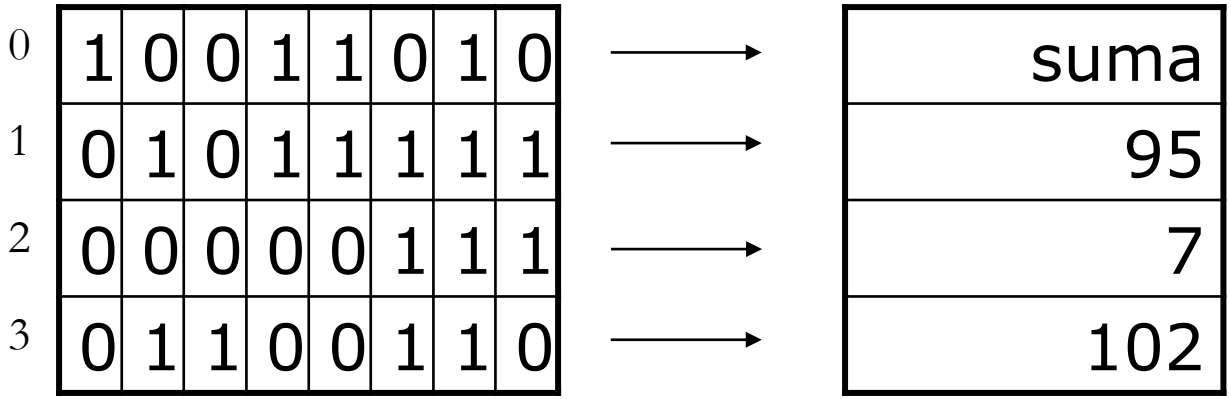
- ¿Qué es programar?
- Breve introducción a la arquitectura de un ordenador
- **Lenguajes de Programación**
- Paradigmas de Programación

Lenguajes de programación

- Programa = datos + algoritmo
- No se puede usar (todavía) lenguaje natural para describir el programa
- ¿Cómo le decimos al ordenador lo que debe hacer?
  - Escribiendo el programa en un lenguaje de programación apropiado (codificando el algoritmo)
- Existen muchos lenguajes de programación (ej. C++, Java, **Python**, etc.)
  - Lenguajes de propósito general vs. Lenguajes específicos

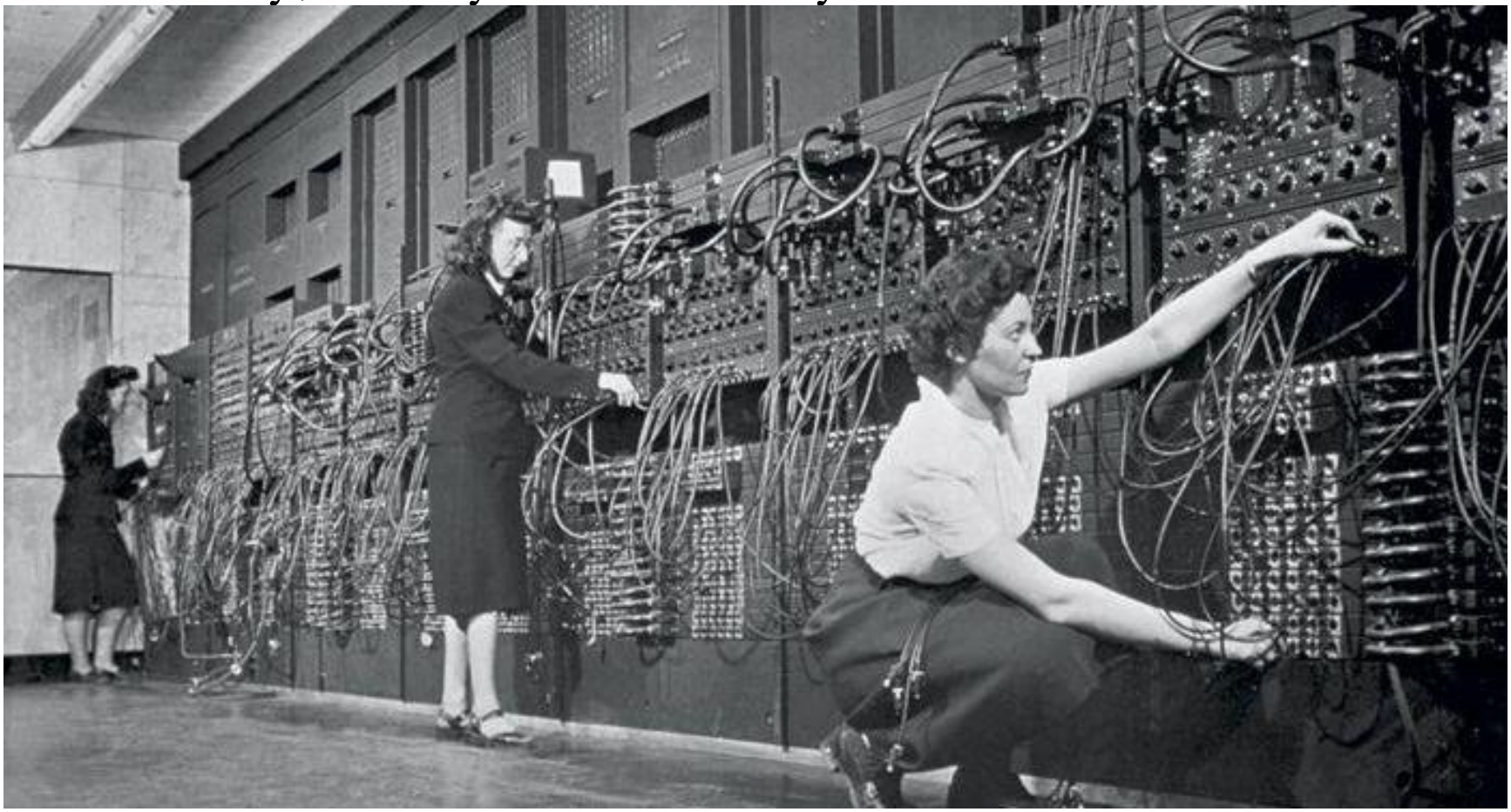
Representación de datos e instrucciones en memoria

- La memoria está compuesta por bits, que solo pueden valer 0 o 1
  - Datos e instrucciones se representan mediante bits
- Los bits se agrupan en bytes (8 bits)
- Cada celda de memoria almacena entre 1 y 8 bytes y tiene una dirección



Las primeras programadoras (ENIAC: 1946)

- Betty Jennings, Betty Snyder, Frances Spence, Kay McNulty, Marlyn Wescoff y Ruth Lichterman



Tipos de lenguajes de programación

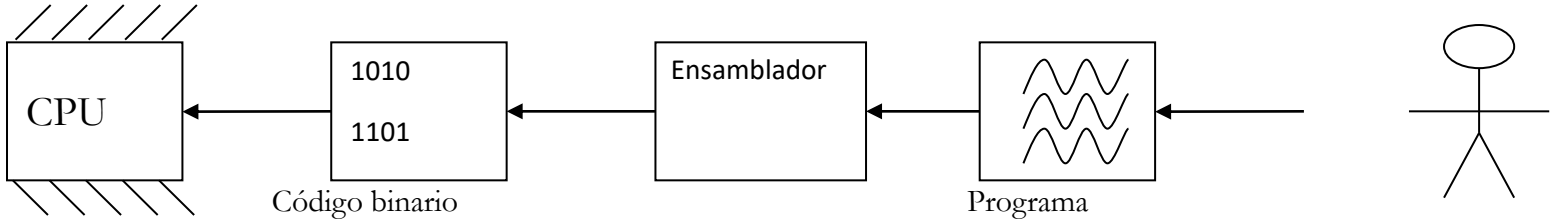
- **Lenguaje binario (código máquina)**
  - 0 y 1
- **Lenguajes de bajo nivel**
  - Instrucciones básicas (mover datos, sumar, etc.)
- **Lenguajes de alto nivel**
  - Más cercanos al lenguaje natural
  - *…aunque tampoco mucho*

Código máquina

- Lenguaje nativo del ordenador, el único que entiende
- Datos e instrucciones se codifican usando 0 y 1
- El más rápido: hablamos al ordenador en su propio lenguaje
- El que menos memoria necesita
- Muy propenso a errores, muy complicado

Lenguajes de bajo nivel

- **Lenguaje Ensamblador:** usa etiquetas (mnemónicos) para las instrucciones y dígitos decimales en lugar de dígitos binarios
- **Ensamblador**: programa que traduce los mnemónicos a sus equivalentes binarios



- No muy intuitivo
- Dependiente del procesador: cada CPU tiene su propio conjunto de instrucciones

Lenguajes de bajo nivel

- Ejemplo de código ensamblador

```
 .model small
 .stack
String1 DB 'Hello World.$'
 .code

 program:
    mov ax, @data
    mov ds, ax
    mov dx, offset String1
    mov ah, 9
    int 21h
    mov  ah,4ch
    int 21h
 end program
```

Lenguajes de alto nivel

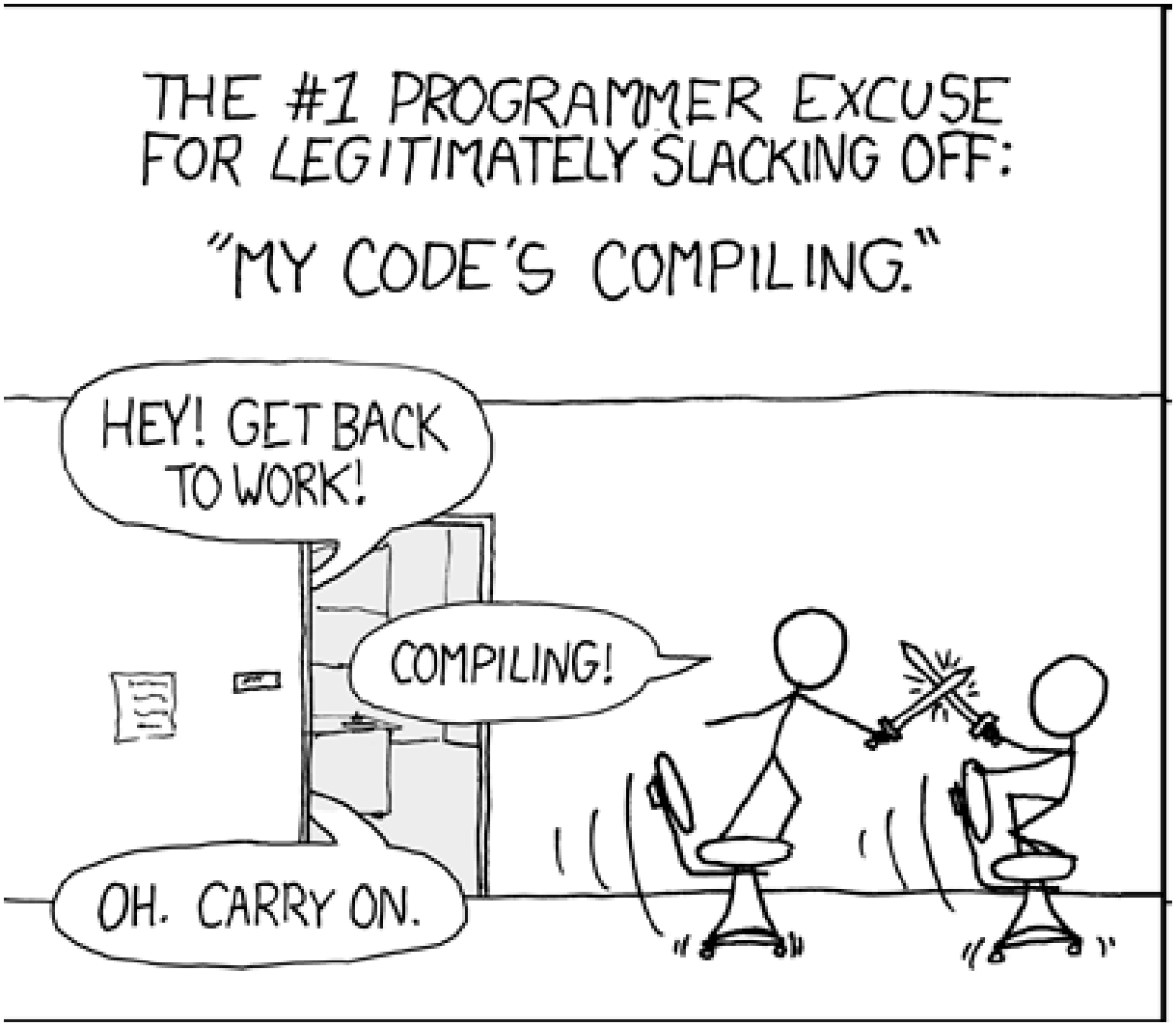
- Intentan acercar el lenguaje de programación al lenguaje humano
- Luego el ordenador se encargará de traducir a código máquina
- Ideal: poder usar lenguaje natural
- Existen más de 300 (unos 2400 con dialectos)
- Los pioneros incluían conceptos como:
  - Variables: no es necesario gestionar directamente la memoria (no hay que saber dónde se guardan los datos)
  - Estructuras de datos complejas
  - Nuevas instrucciones: distintas de las que proporciona el ordenador
  - Librerías: código que resuelve problemas comunes y que podemos usar en nuestros programas
- Historia:

http://manuelpereiragonzalez.blogspot.com/2009/09/historia-de-la-informatica-los.html

Compilación e interpretación

- La traducción de un programa escrito en un lenguaje de programación de alto nivel a binario se puede hacer de dos formas:
- Todo a la vez: compilador
  - El resultado es un programa ejecutable
  - Ejecución más rápida
- Instrucción a instrucción: intérprete
  - Ejecuta aunque haya errores en el código (siempre que la instrucción actual sea correcta)
  - Permite cambios "en caliente"

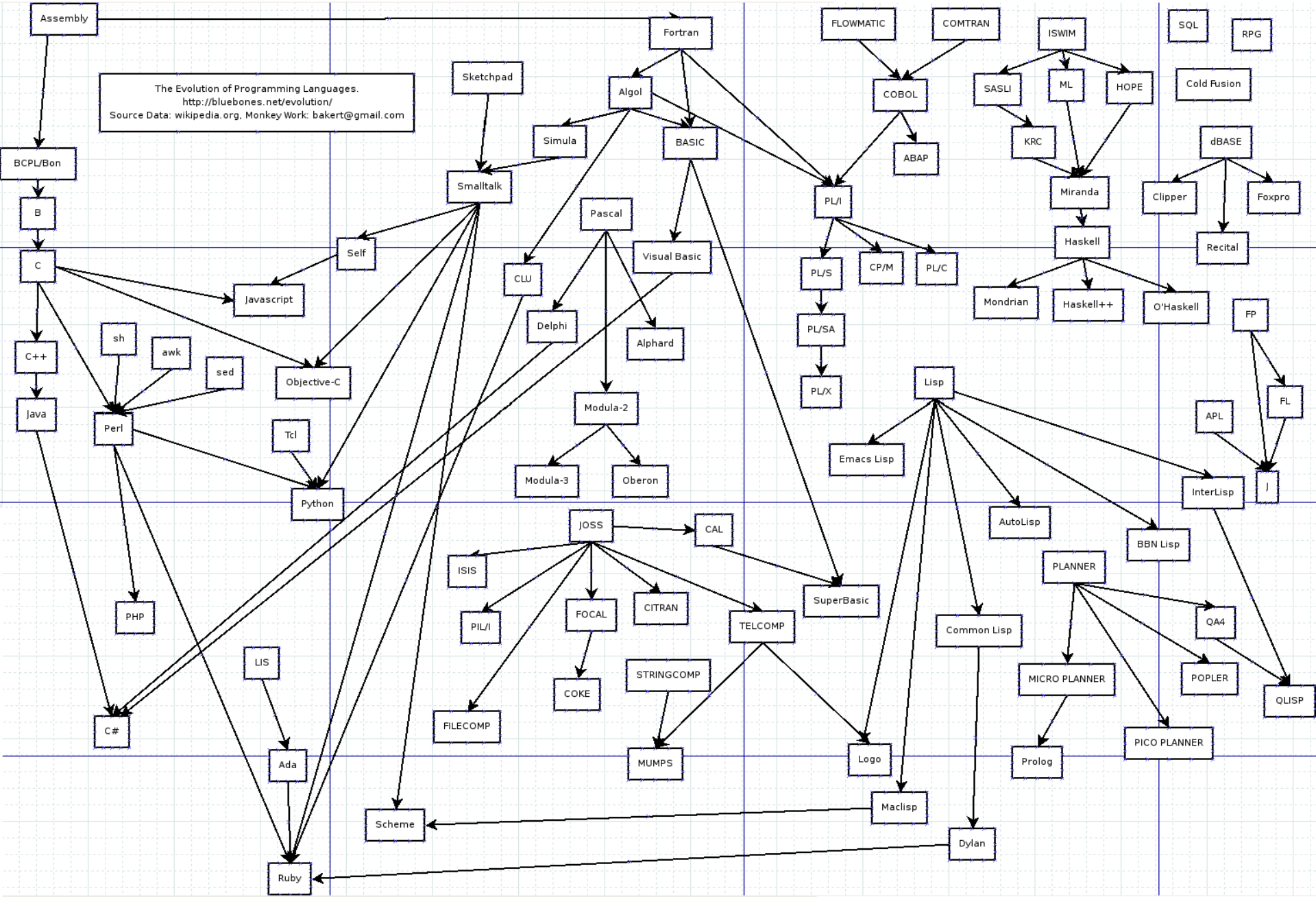
Compilación



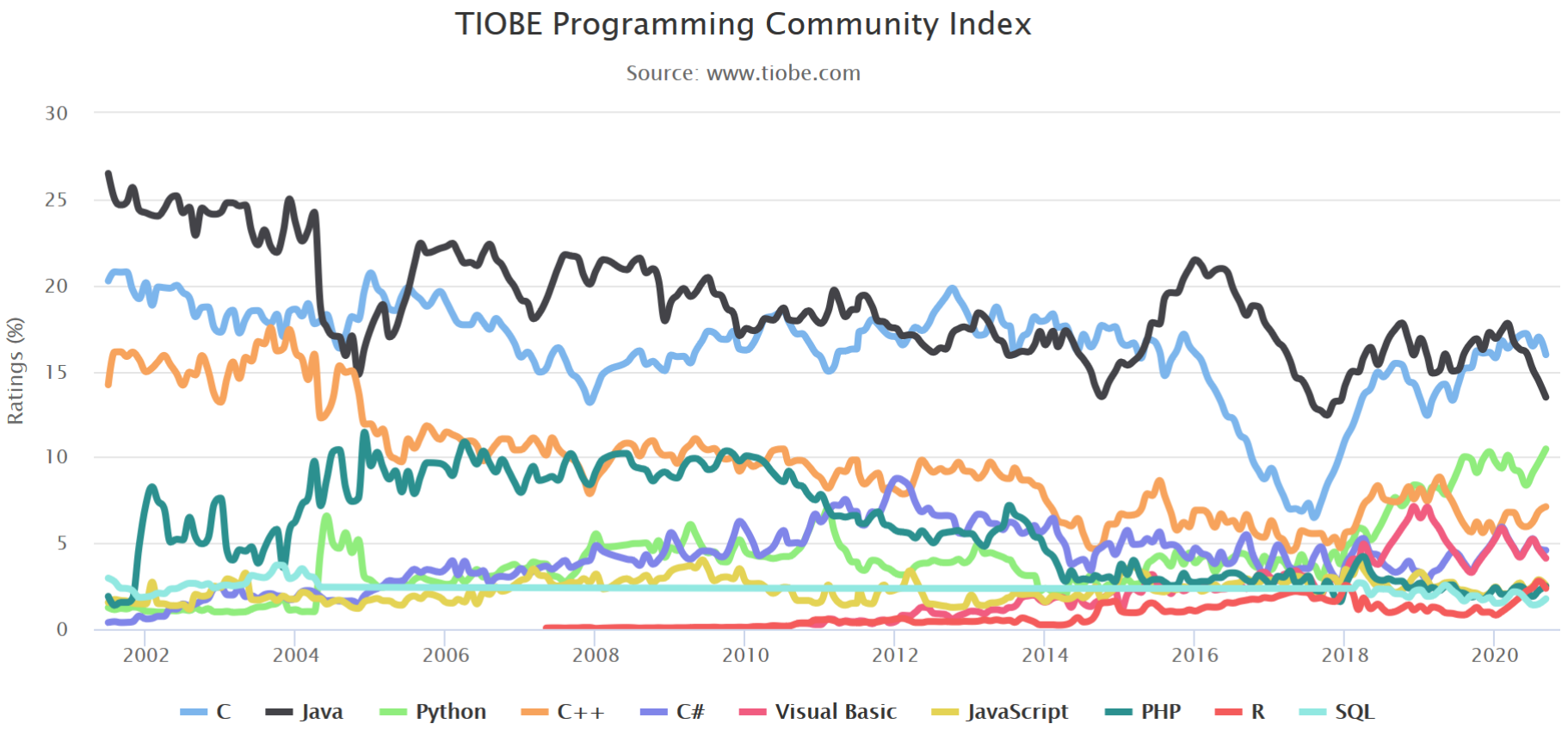
http://xkcd.com/303/

Tiempo de compilación frente a tiempo de ejecución

Evolución de los lenguajes de alto nivel

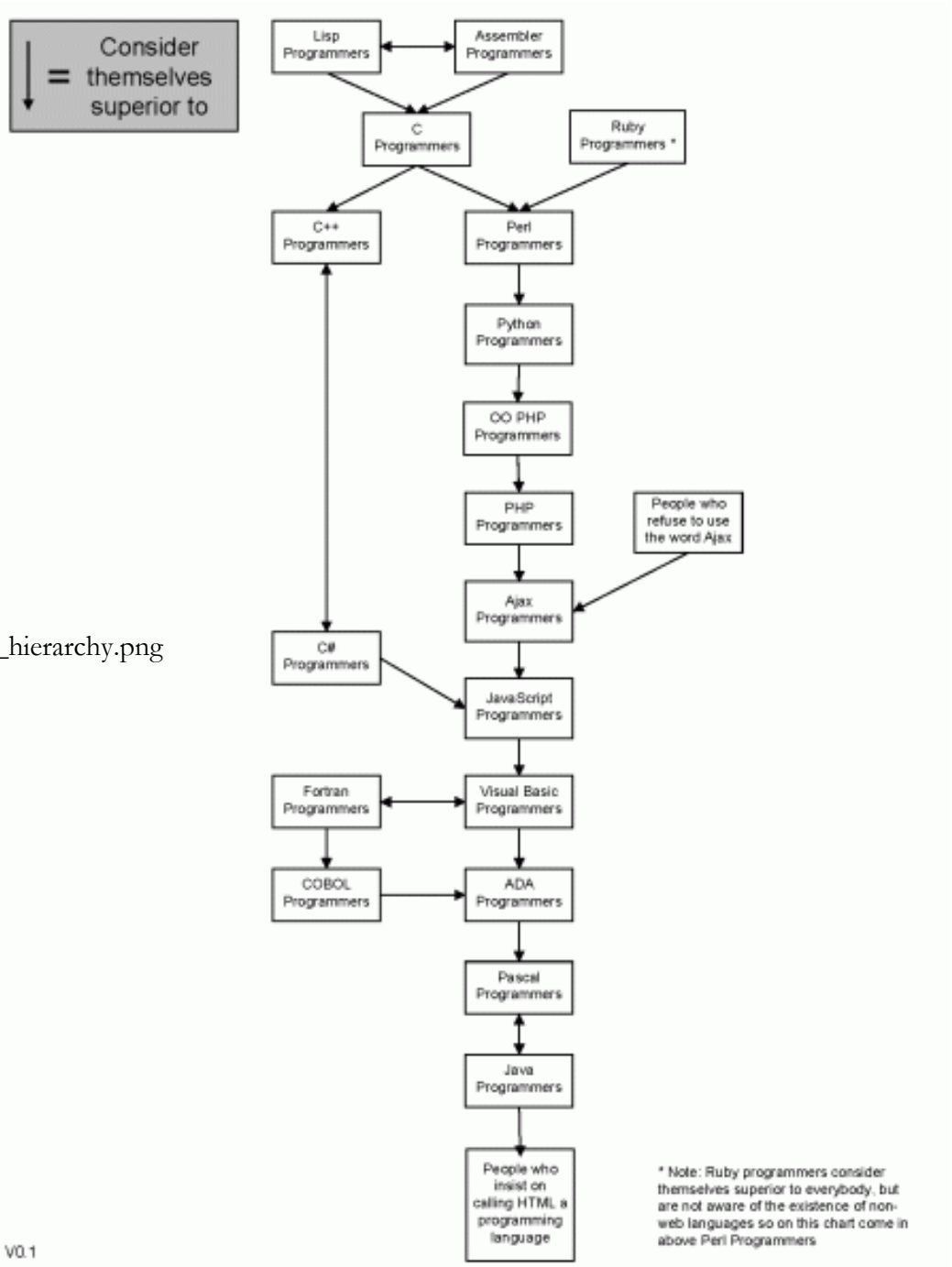


Popularidad de los lenguajes



La jerarquía de los programadores

http://www.hermann-uwe.de/files/images/programmer_hierarchy.png



Índice

- ¿Qué es programar?
- Breve introducción a la arquitectura de un ordenador
- Lenguajes de Programación
- **Paradigmas de Programación**

Paradigmas de programación

- Un **paradigma de programación** es una *filosofía para resolver un problema usando un ordenador*
- **Programación imperativa** (Java, C++, Python, Perl)
  - El programa describe los pasos a seguir para resolver el problema
- **Programación funcional** (Lisp, Erlang, Haskell, F#)
  - El programa describe las transformaciones que deben realizarse en los datos de entrada para obtener la salida deseada
- **Programación lógica** (Prolog)
  - El programa se describe mediante fórmulas lógicas
  - El problema se resuelve usando inferencia lógica
- Ninguno es superior a los demás
- Muchos lenguajes permiten aproximaciones mixtas
  - (ver http://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng.pdf)

Ejemplo: programa para calcular el factorial

Python – f*actorial.python*

```
def factorial(number):
    result = 1
    for ii in range(2, number+1):
        result = result * ii
    print(result)

factorial(42)
```

Haskell – *fac.hs*

```
fac 0 = 1 fac n = n * fac (n-1)
main = print (fac 42)
```

Prolog - *factorial.hs*

```
factorial(0,1).
factorial(N,F) :-
N>0,
N1 is N-1,
factorial(N1,F1),
F is N * F1.
?- factorial(42,X).
```

Aproximaciones a la programación imperativa

- Los lenguajes imperativos modernos son **Estructurados/Procedimentales** u **Orientados a Objetos**
- Los algoritmos suelen ser similares en ambas aproximaciones, la diferencia está en la forma en la que el código se estructura y divide entre diferentes ficheros
- Por lo general la Orientación a Objetos es la mejor aproximación para problemas complejos (¡pero no siempre!)
- La mayoría de los lenguajes Orientados a Objetos permiten usar Programación Estructurada, aunque algunos (como Java) de manera algo forzada
- Dos aproximaciones para aprender programación OO
  - **Empezando por Programación Estructurada**
  - Empezando directamente por Objetos (Objetos primero)

Entornos de desarrollo

- Para programar en un lenguaje solo es necesario:
  - Cualquier editor de textos para escribir el programa
  - Un compilador/intérprete de ese lenguaje para ejecutarlo
- Pero generalmente se utiliza un Entorno de Desarrollo Integrado (IDE)
  - Editor de texto con coloreado
  - Resaltado de errores
  - Autocompletado de nombres y sugerencias
  - Depurador
  - Etc.
- Para Python: Spyder, Thonny, PyCharm…

Resumen

- ¿Qué es programar?
  - Resolver problemas usando un ordenador
  - Necesitamos diseñar un algoritmo…
  - … y codificarlo usando un lenguaje de programación
- Arquitectura de un ordenador
  - CPU
  - Memoria
  - Dispositivos de E/S
  - Bus

- Breve introducción a la programación
  - Código binario
  - Lenguajes de bajo nivel
  - Lenguajes de alto nivel
- Compilación e interpretación
- Paradigmas de programación
  - Programación imperativa
  - Programación funcional
  - Programación lógica
- Estructurada frente a OO
- IDE