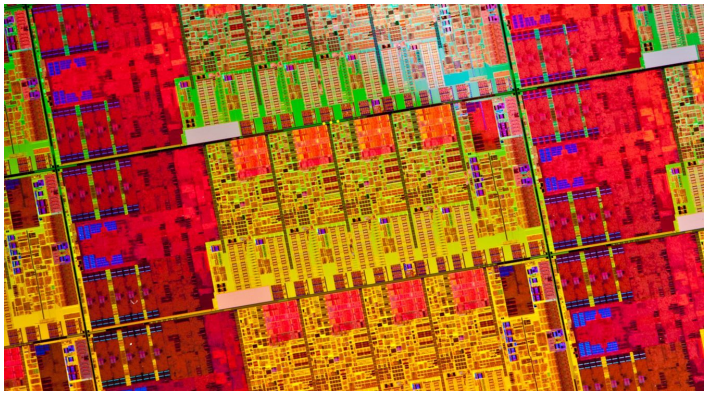
Prohibiendo los bucles. Hacia la programación del siglo XXI.

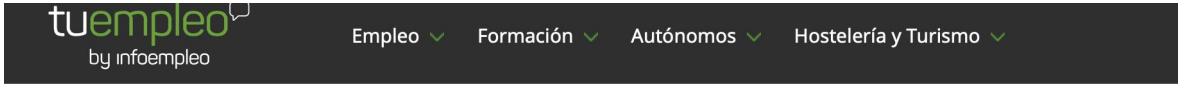
Francisco Fernández de Vega
University of Extremadura
Spain

Programar en el siglo XXI

- La era del “Big Data”
- Aplicaciones intensivas en datos..
- Muchos modelos y librerías para programación paralela/distribuida..
- Infraestructura hardware disponible.:
    - Clusters.
    - Blades.
    - Clouds.
    - Grids.
    - GPUs.
    - Multicores.
    - Maycores...



Programar en el siglo XXI



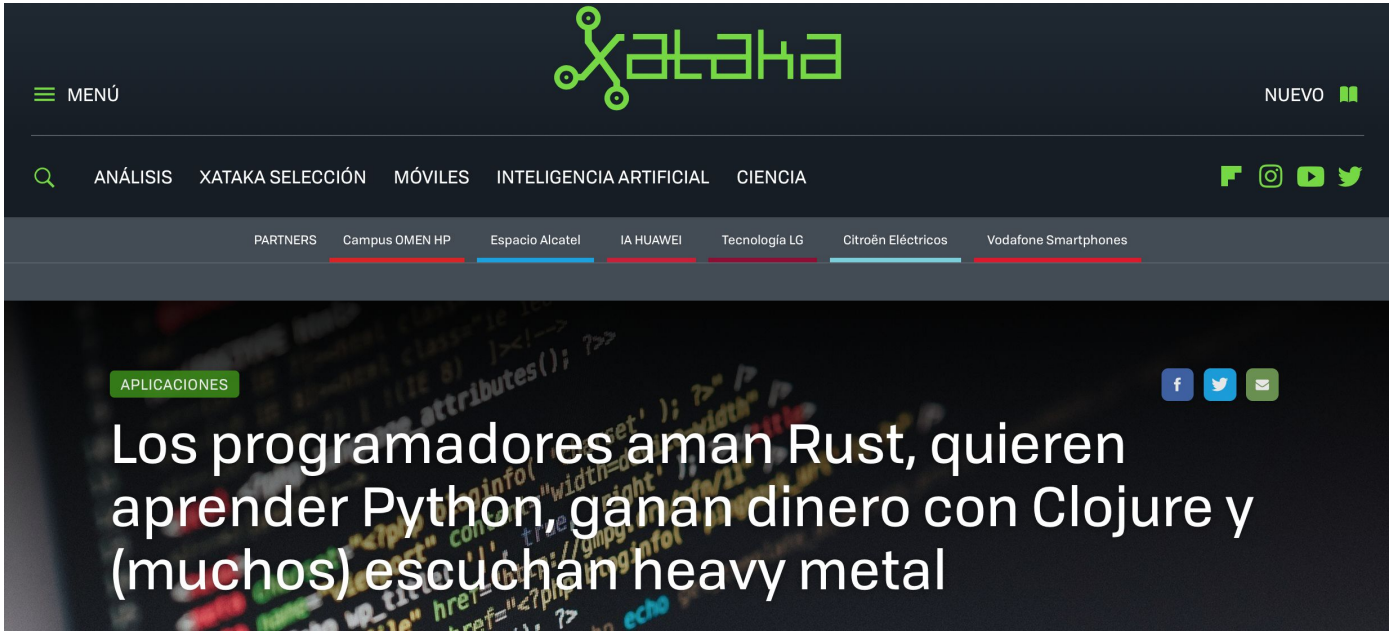
Los lenguajes de programación más demandados

demanda en puestos de trabajo. Este año, **estos son el top ten de los lenguajes más demandados** para cubrir vacantes:

1. Java
2. C
3. Python
4. C++
5. JavaScript
6. C#
7. PHP
8. HTML
9. Ruby
10. Swift

Mención especial merece un lenguaje: R. No está todavía entre los diez primeros según este listado, pero su crecimiento en los dos últimos años ha sido tan espectacular que, si se tienen en cuenta los 12 parámetros del estudio, el IEEE Spectrum lo coloca en sexto lugar, por delante incluso de Java Script. Se trata de un lenguaje orientado a la computación estadística. Cuenta ya con más de 20 años de trayectoria, pero se ha vuelto tan popular porque se está empleando para ciencias de datos y big data.

Programar en el siglo XXI



Programar en el siglo XXI

Programar en el siglo XXI

Metodologías de programación:

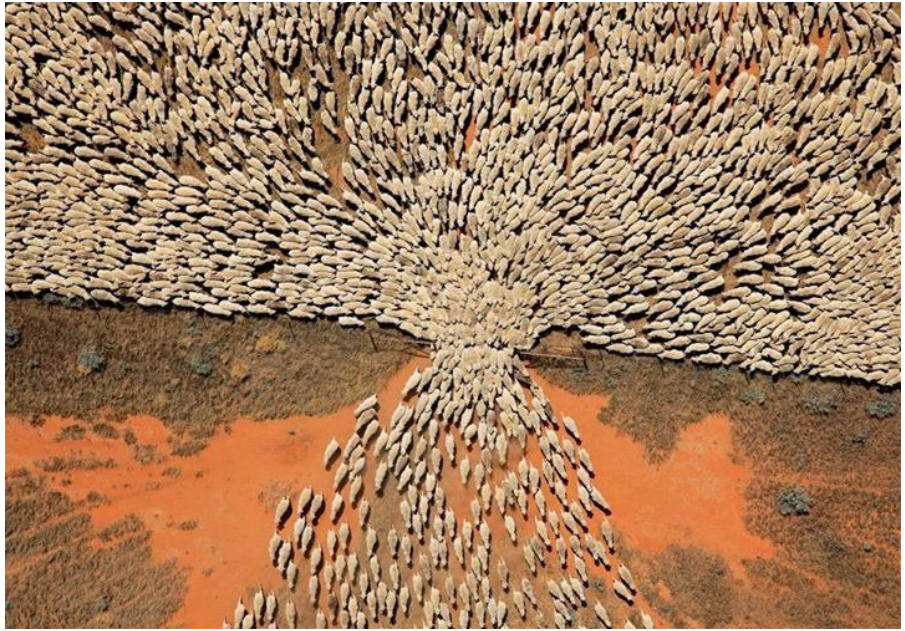
- Programación Estructurada.
- Programación Funcional.
- Programación Orientada a Objetos.

El punto de partida:

¿Estamos enseñando a nuestros estudiantes la programación adecuada para el siglo XXI?

Una observación: OOP es la única metodología enseñada en Extremadura.

El problema de la programación secuencial.



En los primeros cursos del Grado se enseñan programación secuencial (Metodologías OO y lenguaje java).

Programación secuencial basada en bucles para aplicaciones intensivas en datos (SIMD).

Programación Concurrente/paralela/distribuida sólo en los últimos años del grado

¿Es esto lo ideal?

Analicemos la situación

Existen estudios que cuestionan los modelos:

- ¿Qué metodología deberíamos enseñar primero?

Bruce, K. B. (2005). Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list.ACM SIGCSE Bulletin,37(2), 111-117.

- Recursividad antes que bucles si el profesor es competente..

Turbak, F., Royden, C., Stephan, J., & Herbst, J. (1999). Teaching recursion before loops in CS1. Journal of Computing in Small Colleges,14(4),86-101.

La importancia de la recursividad.

Filtros recursivos: La salida del circuito retroalimenta la entrada.

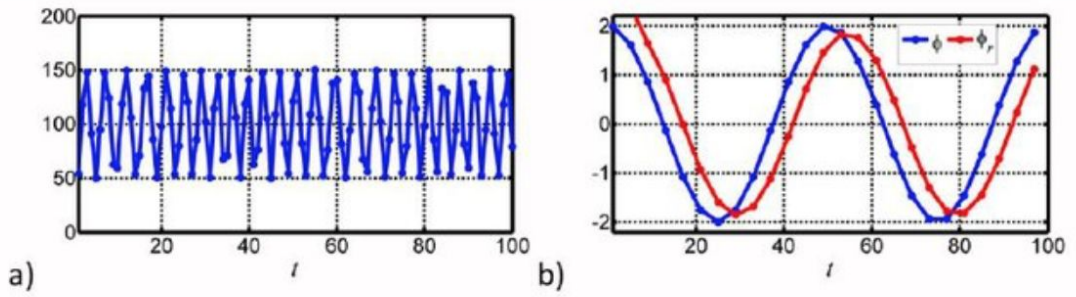


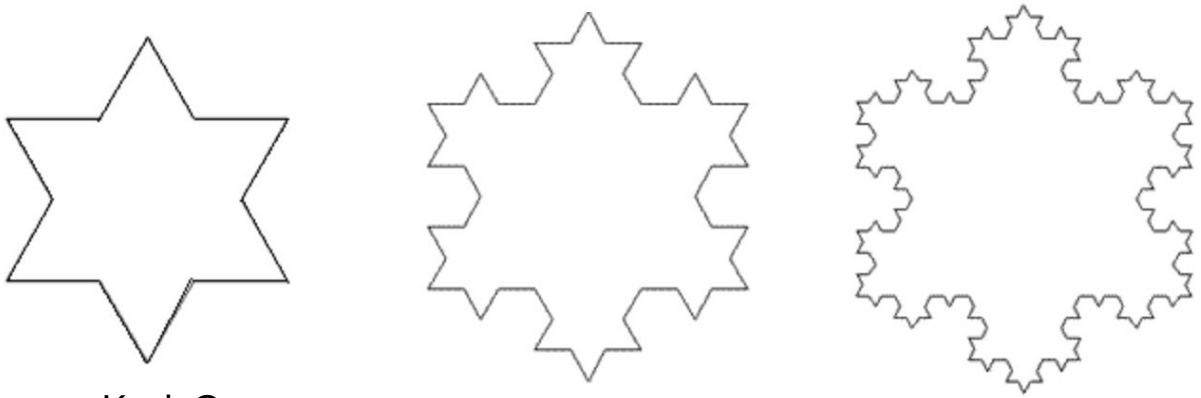
Fig. 2. Demodulation results using the recursive filter of Eq. (18). a) input signal, b) demodulation results (red) and actual phase (blue). In this figure there is a marker every four samples to make easier the observation of the 4 samples delay between both signals.

Role of the filter phase in phase sampling interferometry. J.A. Quiroga et al, Optics Express 2011.

La importancia de la recursividad

La recursividad más allá de la programación

Fractales: Objetos autosimilares a cualquier escala. Producidos de forma iterativa.



Koch Curve.

Si los fractales son recursivos...

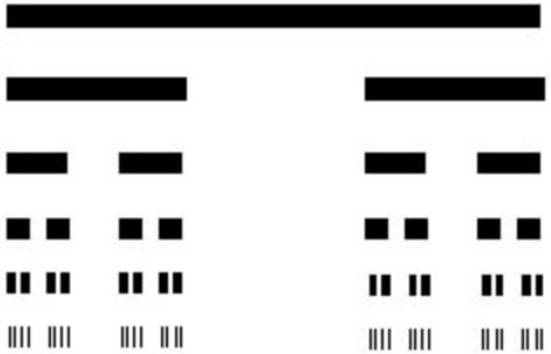
Fractales en:

- Música: Bach 4 suites para chelo.

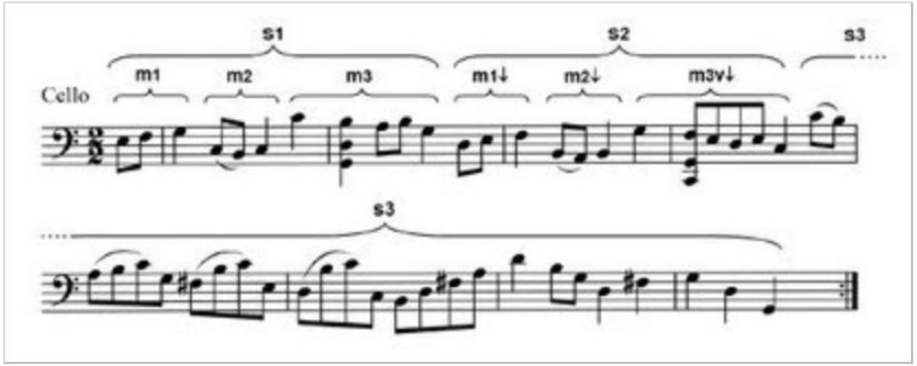
Si los fractales son recursivos...

Fractales en:

- Música: [Bach 4 suites para chelo.]



El “peine” de Cantor.



Analysis of the first 16 measures of the Bourrée from Bach's Suite No. 3. Courtesy of Harlan Brothers.

The mathematical tourist.

Fractales en la naturaleza

¿Recursividad antes que bucles?

Razones pedagógicas: Divide y vencerás.

Cualquier problema repetitivo puede resolverse mediante recursividad, pero en algunos casos se requiere si no se usa ésta, y en su lugar se utilizan bucles, es necesario utilizar una estructura de pila adicional (problemas en que se requiere memoria).

La recursividad de “cola” evita los desbordamientos.....

Pero, ¿qué prefieren nuestros estudiantes y los programadores profesionales?

Analizando la situación en Extremadura

Lanzamos una encuesta sobre técnicas de programación:

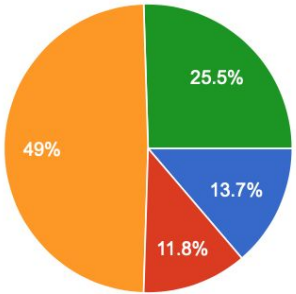
- Una factoría software.
- Estudiantes del último año del Grado en Informática.

Las principales preguntas versaban sobre:

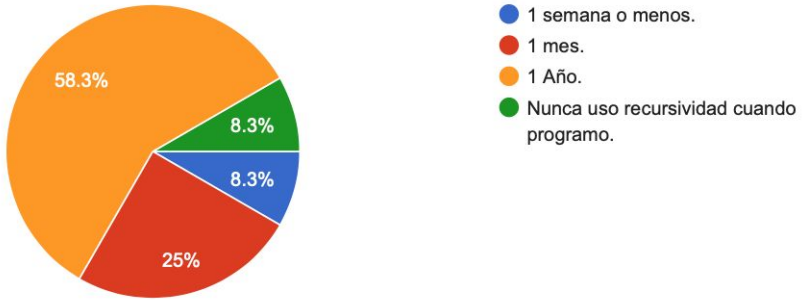
- Metodologías de programación.
- Recursividad..
- Utilización de programación paralela.

¿Cuánto tiempo hace que usaste por última vez una función recursiva?

Programadores profesionales

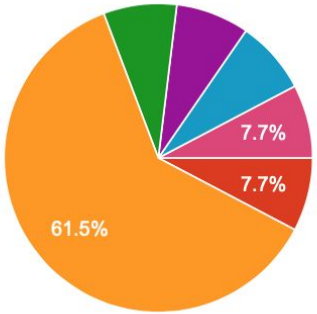


Estudiantes de último curso

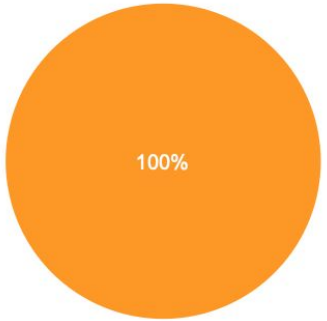


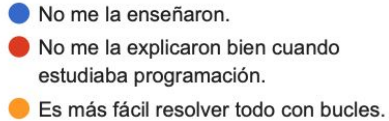
¿Porqué nunca utilizas recursividad?

Programadores profesionales



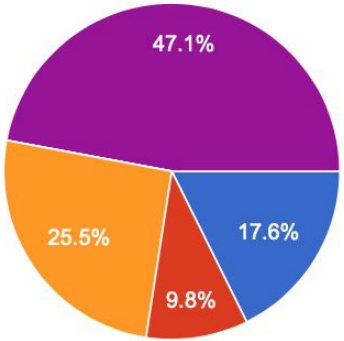
Estudiantes de último curso.



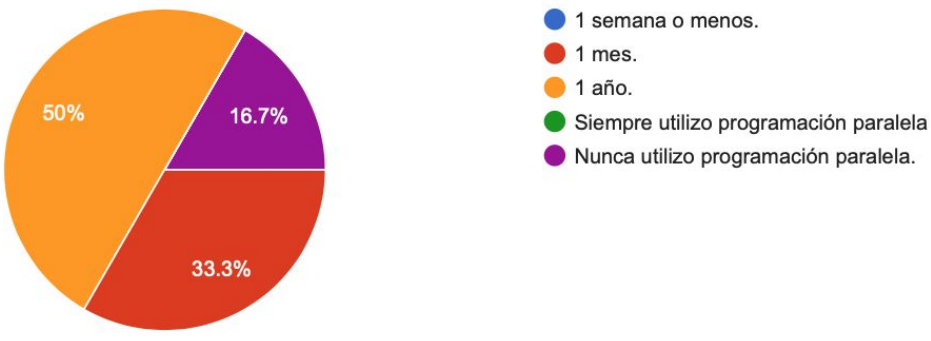


¿Cuánto hace que utilizaste por última vez programación paralela?

Programadores profesionales

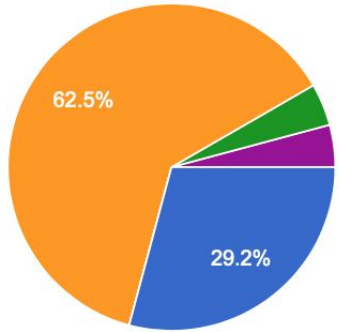


Estudiantes de último curso.

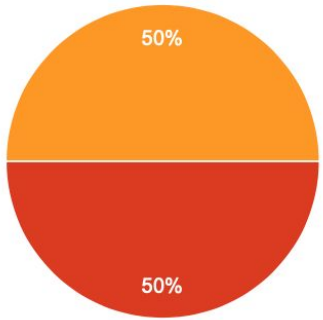


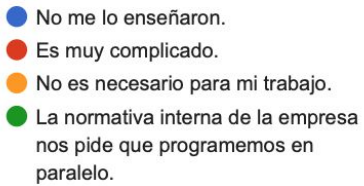
¿Porqué nunca utilizas la programación paralela?

Programadores profesionales



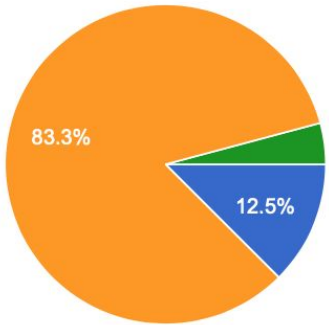
Estudiantes de último curso.



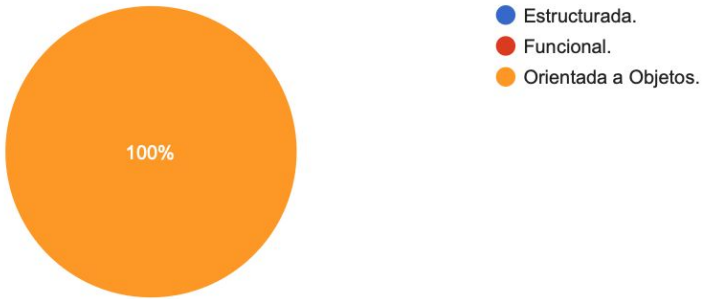


¿Qué metodología de programación utilizas?

Programadores profesionales



Estudiantes de último curso.



La programación del siglo XXI, resumiendo:

- Estudiantes y profesionales evitan la recursividad….
- ...y rechazan la programación paralela..
- Les cuesta aprender y aplicar ambos conceptos..

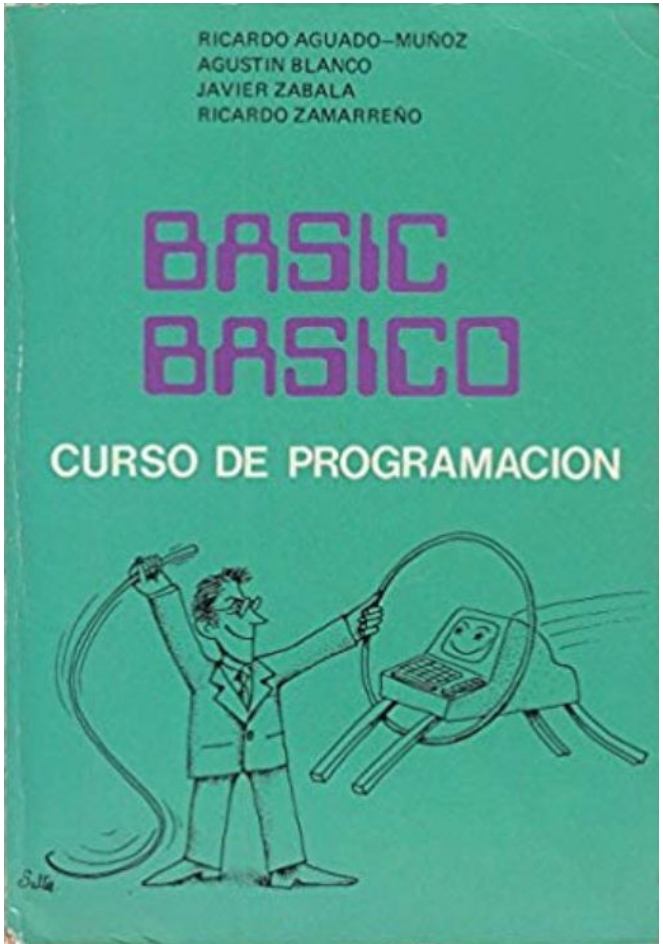
¿Habrá alguna relación entre esta situación y la forma en que aprendieron a programar?

¿Alguna conexión entre estas dos cuestiones: programación paralela y evitar la recursividad?

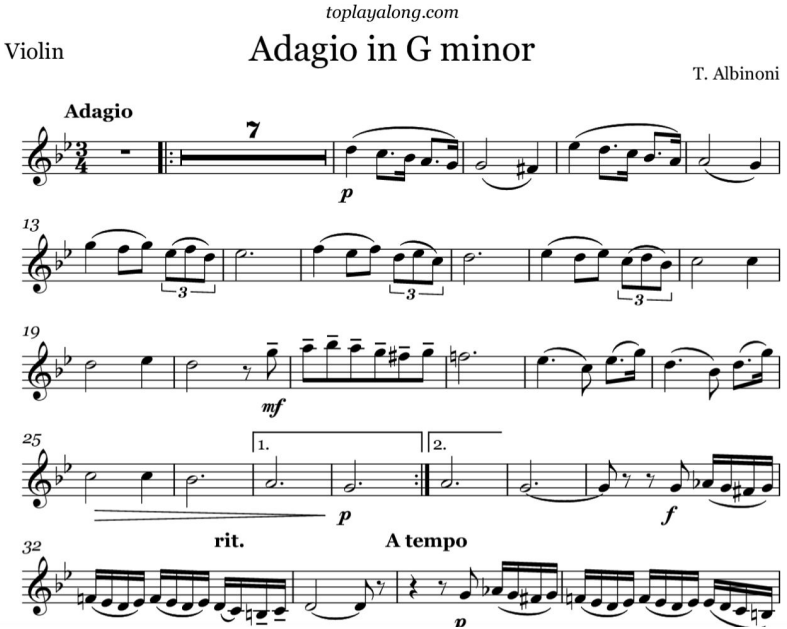
¿Qué podemos hacer?

Podemos aprender del pasado para mejorar el futuro.

Back to the future



Back to the future

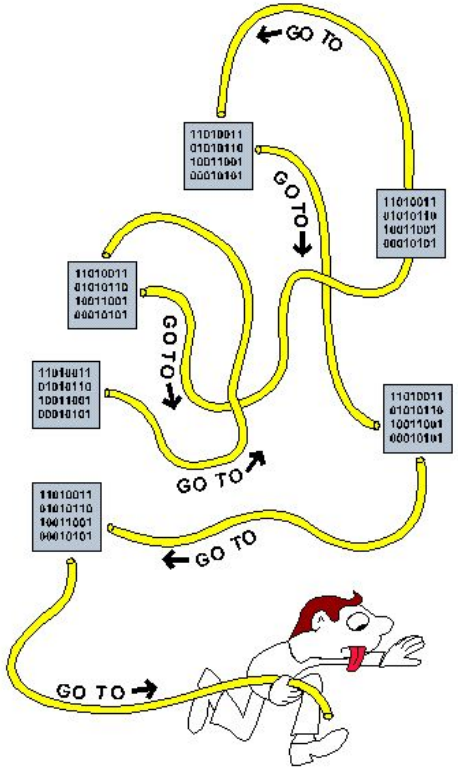


Back to the future

¿Qué podemos hacer?

En los años setenta, las metodologías que disponíamos:

- Programación basada en saltos “go-to” (Código Espagueti).
- Programación estructurada..



¿Qué podemos hacer?

Structured Programming with go to Statements

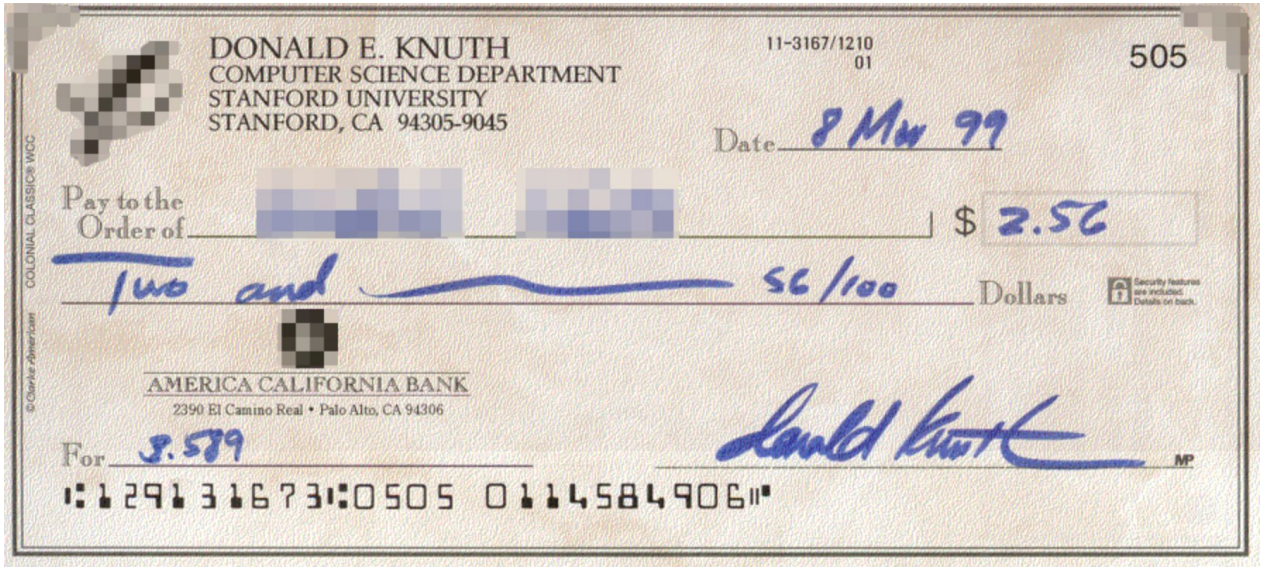
DONALD E. KNUTH

Stanford University, Stanford, California 94305

- Este trabajo (1974) se fijaba en dos cuestiones:
  - Mejorar la sintaxis de las iteracciones.
  - Diseñar una metodología para el desarrollo de programas..

Donald E. Knuth

Premio BBVA Fronteras del conocimiento 2010,



Wikipedia

¿Qué podemos hacer?

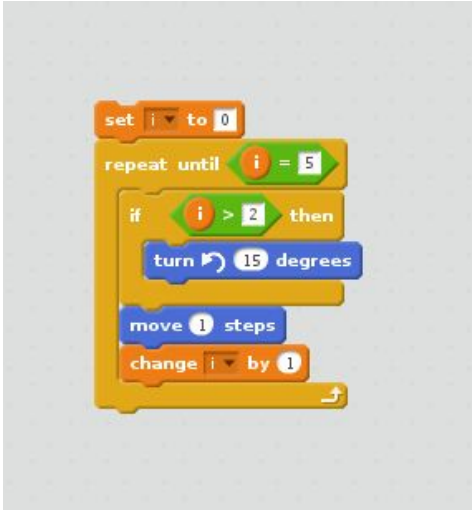
Structured Programming with go to Statements

DONALD E. KNUTH

Stanford University, Stanford, California 94305

- Se refiere a una conferencia de Dijkstra “El programador humilde” y al libro “Programación estructurada” (Dahl, Dijkstra, Hoare).
- Discute si las sentencias go-to tienen una estructura sintáctica fácil de ver en el código; Pero en cualquier caso no culpabiliza a los programadores que la utilicen.
- No obstante, reconoce que “go-to es casi nunca la mejor alternativa disponible en los lenguajes de programación actuales”.

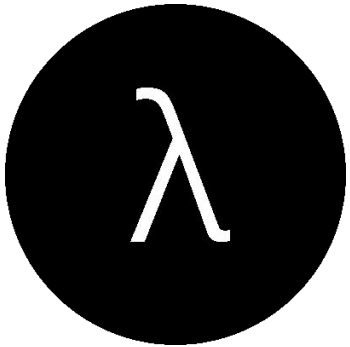
Lo que se enseña en la actualidad



Prog. Orientada a Objetos (y estructurada):

- Modelos basados en los bucles para que los niños aprendan programación (scratch, por ejemplo).
- Modelos de programación secuenciales (iterativos)..
- Una vez el modelo se asume, el programador tiene a aplicarlo por defecto, aunque se ofrezcan otras alternativas.
- El programador no tiene en cuenta las características del hardware disponible ni las posibles dependencias de datos.

Pero existen otros mundos



FUNCTIONAL PROGRAMMING LANGUAGES

ERLANG

elixir

Haskell

OCaml

Scala

Clojure

El problema del bucle.

¿Qué sucede cuando pedimos a nuestros estudiantes OOP que resuelvan los problemas siguientes?…

1.- Sumar uno a cada posición de un array. a[i]=a[i]+1.

2.- Acumular secuencialmente todas las posiciones de un array. a[i+1]=a[i+1]+[ai].

Tienden a producir soluciones como éstas:

```
for (i=0;i<n;i++)
  a[i]++;
```

```
for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

El problema del bucle.

Intentemos ahora resolver los mismos problemas en paralelo, utilizando cualquier librería de programación paralela que paralelice los bucles:

1.- Sumar uno a cada posición del array. a[i]=a[i]+1.

2.- Acumular secuencialmente las posiciones del array. a[i+1]=a[i+1]+[ai].

Lanzamos en paralelo los bucles for:

```
for (i=0;i<n;i++)
  a[i]++;
```

```
for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

¿Dónde está el problema?

Los estudiantes no tienen práctica en apreciar las dependencias de datos.

La razón es que han aprendido programación SECUENCIAL para resolver ambos problemas:

```
for (i=0;i<n;i++)
  a[i]++;

for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

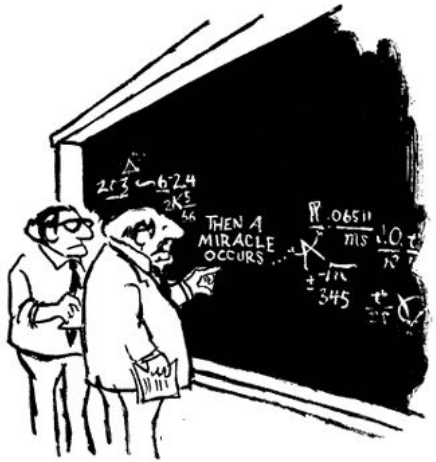
```
for (i=0;i<n;i++)
  a[i]++;

for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

FOR lanzados en paralelo.

Funcional vs Metodologías Orientadas a Objeto.

Programación Funcional:

- Programación declarativa.
- Basada en la recursividad.
- Funciones Puras.
- Funciones de alto orden (map).
- Macros: transformaciones código-datos..
- Cálculo Lambda (A. Church).



To be or not to be, that is the question

SPANISH		ENGLISH	
SER	Yo soy Español	TO BE	I'm Spanish.
ESTAR	Yo estoy en Dubai	TO BE (in a place)	I'm in Dubai.

To be or not to be, that is the question

Problem to be solved	Data involved	Nature of the problem
a[i]=a[i]+1.	No dependencies	Parallel.
a[i+1]=a[i+1]+[ai]	Data dependencies.	Sequential.

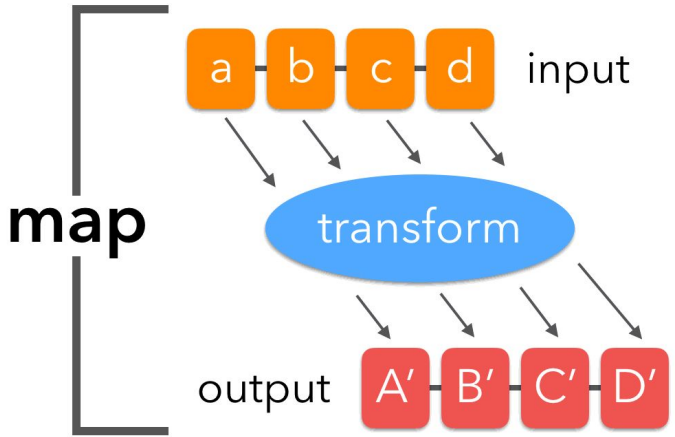
To be or not to be, that is the question

Data involved	**Loop based** programming (OOP, Structured)	**Nature of the problem.**
No dependencies	`for (i=0;i<n;i++)` `  a[i]++;`	Parallel (solved sequentially).
Data dependencies.	`for (i=1;i<n;i++)` `  a[i]=a[i]+a[i-1];`	Sequential.

Propuesta: Prohibir los bucles.

¿Cómo resolvemos tareas repetitivas sin bucles?

- Funciones recursivas.
- Funciones de alto orden- MAP: se aplica a todos los elementos a la vez.



To be or not to be, that is the question

Datos

Programación basada en **Bucles** (OOP, Structured)

Functional + Prohibición de bucles

Independientes.

```
for (i=0;i<n;i++)
  a[i]++;
```

MAP

Con dependencias.

```
for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

Recursividad

Funcional + prohibición de bucles

Dos opciones para sumar uno a cada elemento de una lista:

SOLUCIÓN RECURSIVA

```
(defun solution (a)
   (if (null a)
       a
     (cons (1+ (car a))
           (solution (cdr a)))))
```

SOLUCIÓN MAP

```
(mapcar '1+ a)
```

Funcional + prohibición de bucles

Para el segundo problema

SOLUCIÓN RECURSIVA

```
(defun solution (a)
  (if (null (cdr a))
      a
    (cons (car a)
       (solution
           (cons (+ (car a)
                    (cadr a))
                 (cddr a))))))
```

SOLUCIÓN MAP

```
(mapcar '+ (cons '0 a) (append a '(0)))
```

Aplicando la metodología

Jóvenes estudiantes de las Escuelas Municipales de Jóvenes Científicos. 400 estudiantes en 20 localidades.

http://www.uexfundacion.es/jovenescientificos/

Aplicando la metodología

Escuela Municipal de jóvenescientíficos

http://www.uexfundacion.es/jovenescientificos/

Jóvenes estudiantes de las Escuelas Municipales de Jóvenes Científicos. 400 estudiantes en 20 localidades.

TABLE VI
PERCENTAGE OF POPULATION WITH (OR PURSUING) COLLEGE DEGREES

Where	University	STEM degrees	STEM at H.S.
Spain	40%	26%	41%
Extremadura	20%	26%	45%
MSYS average	48%	47%	57%
MSYS everybody-pays	47%	75%	90%

http://www.uexfundacion.es/jovenescientificos/

Aplicando la metodología

Después de 40 horas de clase de COMMMON LISP, en que han aprendido los principios de la recursividad y las funciones de alto orden, les planteamos los siguientes problemas:

- Calcular el volumen de una serie de cilindros. Los radios y alturas de los mismos suministrados en dos listas..
- El camino del borracho: Un borracho camina por una calle estrecha con riesgo de chocar con alguna de las paredes de la misma. Cada paso al azar que da, a derecha o izquierda, se acerca a una de las dos paredes y se aleja de la contraria. Después de un número de pasos al azar, se trata de saber si habrá chocado con alguna pared. Se suministra una lista de pasos, 0-derecha, 1-izquierda, por ejemplo (1 0 0 1 0 1 1 1 0 0 1).

Aplicando la metodología

http://www.uexfundacion.es/jovenescientificos/

Los estudiantes fueron capaces de generar estas dos soluciones::

```
(defun volume (r-list h-list)
  (mapcar 'area r-list h-list))

(defun area (r)
  (* 3.14159 r r))
```

```
(defun drunkards-walk (steps-list)
  (check-the-random-walk
     steps-list 0 )) ;begin at position '0

(defun check-the-random-walk (steps-list n)
  (if (null steps-list)
      'congratulations
      (if (or (= n 5)(= n -5))
         'bad-luck
        (if (= 0 (car steps-list))
           (check-the-random-walk
             (cdr steps-list) (-1 n))
          (check-the-random-walk
             (cdr steps-list) (+1 n))))))
```

¿Puedo hoy día aplicar LISP en paralelo?

Bordeaux Threads

Portable shared-state concurrency for Common Lisp

Based on an original proposal by Dan Barlow (Bordeaux-MP) this library is meant to make writing portable multi-threa

Read the current API documentation.

Supports all major Common Lisp implementations: SBCL, CCL, Lispworks, Allegro, ABCL, ECL, Clisp.
The MKCL, Corman, MCL and Scieneer backends are not tested frequently(if ever) and might not work.

For discussion, use the mailing list bordeaux-threads-devel or the #lisp IRC channel on Freenode.

IEEE Xplore® Digital Library

Institutional Sign In

Browse | My Settings | Get Help | Subscribe

All | Enter keywords or phrases (Note: Searches metadata only by default. A search for 'smart grid' = 'smart AN

Conferences > 2018 IEEE International Confe...

And Now for Something Completely Different: Running Lisp on GPUs

4 Author(s) Tim Süß ; Nils Döring ; André Brinkmann ; Lars Nagel View All Authors

lparallel.org

Inbox (5) - fcofdezdevega@g... | Conferencia Madrid - Google... | Diario Oficial de Extremadura | My Drive - Google Drive | Conferencia Madrid - Googl

Lisp in Parallel : lparallel

(ql:quickload :lparal

Home Overview Download Kernel Handling Promises Cognates *pmap* *preduce* *defpun* Ptrees Benchmarks

lparallel-2.8.0 released

Report a bug or requ
lparallel

Conclusiones

Modelo Funcional más apropiado para el siglo XXI.

Los bucles deberían prohibirse: las funciones MAP se usarán en tareas paralelas con datos sin dependencias, y la recursividad en tareas repetitivas que requieran un orden.

El código producido, es paralelo“per se”.

Gracias.

Preguntas.