Programación Declarativa

Curso 2019-20

INTRODUCCIÓN

Un poco de Historia

Hace muchos años...

1930's: Nace la Teoría de la Computabilidad

Nociones teóricas, no había ordenadores

- Máquinas de Turing

  dispositivo de cómputo $\rightsquigarrow$ arquitectura Von-Neumann **programación imperativa**

- λ-cálculo (Church), funciones recursivas (Gödel, Kleene)

  formalismos abstractos $\rightsquigarrow$ definición de funciones **programación funcional**

Hace aún más años . . .

- s. IV a.C - s. XIX: Lógica (más o menos formal)
  - Soporte de los aspectos deductivos del conocimiento humano
- 1870-1920: Lógica primer orden
  - Soporte (casi) universal del razonamiento matemático
- 1950: Lógica de Horn
  - Fragmento de la lógica de primer orden fácilmente mecanizable
- 1970-80's: $\rightsquigarrow$ **Programación lógica**
  - Se empieza a explotar el valor computacional de las teorías lógicas y los procesos deductivos
  - Surge Prolog, un lenguaje de programación basado directamente en la lógica de Horn

Programación imperativa vs. declarativa

Alan J. Robinson (uno de los padres de la programación lógica)

La visión de la computación a lo Turing/VonNeumann pone énfasis en la actividad de los cómputos ('cómo'), mientras que el punto de vista declarativo pone énfasis en el resultado de los cómputos ('qué')

Sir Anthony Hoare – Premio Turing 1980 *(primer Doctor Honoris Causa de UCM en Informática, mayo 2013)*

. . . There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

Richard OKeefe – guru de Prolog, autor de *The Craft of Prolog*

Elegance is not optional. *There is no tension between writing a beautiful program and writing an efficient program. If your code is ugly, the chances are that you either don't understand your problem or you don't understand your programming language, and in neither case does your code stand much chance of being efficient. In order to ensure that your program is efficient, you need to know what it is doing, and if your code is ugly, you will find it hard to analyse.*

¿Qué hace este código?

```
procedure hazalgo(l,r:index);
var i,j:index; x,w:item
begin
  i := l; j := r;
  x := a[(l+r) div 2];
  repeat
   while a[i] < x do i := i+1;
   while x < a[j] do j := j-1;
   if i <= j then
   begin
    w := a[i]; a[i] := a[j]; a[j] := w;
    i := i+1; j := j-1
   end
  until i > j;
  if l < j then hazalgo(l,j);
  if i < r then hazalgo(i,r);
end
```

¿Qué hace este código?

```
procedure quicksort(l,r:index);
var i,j:index; x,w:item
begin
  i := l; j := r;
  x := a[(l+r) div 2];
  repeat
   while a[i] < x do i := i+1;
   while x < a[j] do j := j-1;
   if i <= j then
   begin
    w := a[i]; a[i] := a[j]; a[j] := w;
    i := i+1; j := j-1
   end
  until i > j;
  if l < j then quicksort(l,j);
  if i < r then quicksort(i,r);
end
```

Versión declarativa

```
qsort   []      =   []
qsort   (x:l)   =
  qsort [y | y <- l, y < x]
        ++ [x] ++
  qsort [y | y <- l, y > x]
```

¿Qué hace este código?

```
procedure quicksort(l,r:index);
var i,j:index; x,w:item
begin
  i := l; j := r;
  x := a[(l+r) div 2];
  repeat
   while a[i] < x do i := i+1;
   while x < a[j] do j := j-1;
   if i <= j then
   begin
    w := a[i]; a[i] := a[j]; a[j] := w;
    i := i+1; j := j-1
   end
  until i > j;
  if l < j then quicksort(l,j);
  if i < r then quicksort(i,r);
end
```

Versión declarativa

```
qsort   []      =   []
qsort   (x:l)   =
  qsort [y | y <- l, y < x]
        ++ [x] ++
  qsort [y | y <- l, y > x]
```

- No hay noción de variable asignable
- No hay noción de cambio de estado
- Recursión en lugar de iteración

¿Qué hace este código?

```
procedure quicksort(l,r:index);
var i,j:index; x,w:item
begin
  i := l; j := r;
  x := a[(l+r) div 2];
  repeat
   while a[i] < x do i := i+1;
   while x < a[j] do j := j-1;
   if i <= j then
   begin
    w := a[i]; a[i] := a[j]; a[j] := w;
    i := i+1; j := j-1
   end
  until i > j;
  if l < j then quicksort(l,j);
  if i < r then quicksort(i,r);
end
```

Versión declarativa

```
qsort   []      =   []
qsort   (x:l)   =
  qsort [y | y <- l, y < x]
        ++ [x] ++
  qsort [y | y <- l, y > x]
```

- No hay noción de variable asignable
- No hay noción de cambio de estado
- Recursión en lugar de iteración