

# PROGRAMACIÓN DECLARATIVA

## Introducción

Curso 2018/19

# Programación Imperativa vs. Declarativa

- Programación imperativa
  - La visión de la computación a lo Turing/Von Neumann pone énfasis en la actividad de los cálculos (*cómo*)
  - Dispositivo de cálculo -> arquitectura Von-Neumann
- Programación Declarativa
  - El punto de vista declarativo pone énfasis en el resultado de los cálculos (*qué*)
  - Se basa en formalismos abstractos
  - Paradigmas funcional y lógico

# Ejemplo:

## Suma de dos números naturales

- En Imperativo

```
proc suma(x, y, z)
  z := x;
  while y > 0 do
    {z := z + 1;
     y := y - 1}
  end while
end proc
```

%  $z = x + y$

- En Declarativo

```
suma(X, 0, X).
suma(X, suc(Y), suc(Z)) :- suma(X, Y, Z).
```

%  $\text{suma}(X, Y, Z) \Leftrightarrow Z = X + Y$

```
suma x y
  | y == 0      = x
  | otherwise   = 1 + suma x (y-1)
```

%  $\text{suma}(x, y) = x + y$

# Paradigmas funcional y lógico

- Programación funcional
  - Formalismos:  $\lambda$ -cálculo (Church), funciones recursivas (Gödel, Kleene)
  - Programas: Definición de funciones
  - Cómputos: Evaluación de expresiones (atómicas o aplicación de funciones a argumentos)
- Programación lógica
  - Formalismos: Lógica de Cláusulas de Horn y Resolución
  - Programas: Definición de relaciones
  - Computación: Deducciones para resolver objetivos (fórmulas de la lógica)

# Lenguajes de programación funcionales y lógicos

- Programación funcional
  - **Haskell**, Lisp, Scheme, ML, Caml, OCaml, Clean, Erlang, Scala . . .
  - Características que los distinguen:
    - Evaluación impaciente/ evaluación perezosa
    - Tipado estático/ tipado dinámico
    - Concurrencia, Orientación objetos...
- Programación lógica
  - **Prolog**, Oz, Mercury,  $\lambda$ -Prolog, Curry...
  - Características que los distinguen:
    - Combinación con otros paradigmas
    - Introducción de tipos
    - Orden superior

## Ejemplo:

### Suma de los n primeros números naturales (I)

- En Imperativo (JAVA)

```
int total = 0;
```

```
for (int cont = 1; cont <= n; cont ++)
```

```
    total = total + cont;
```

Cómputo para n = 5

```
total = 0;
```

```
cont = 1; total =1;
```

```
cont = 2; total =3;
```

```
cont = 3; total =6;
```

```
cont = 4; total =10;
```

```
cont = 5; total =15;
```

## Ejemplo:

### Suma de los n primeros números naturales (II)

- En funcional (Haskell)

`sum [1..5]`

Cómputo para  $n = 5$

`sum [1,2,3,4,5]`      aplicar definición de `[ .. ]`

`= 1 + 2 + 3 + 4 + 5`      aplicar `sum`

`= 15`      aplicar `+`

- En lógico (Prolog)

`sumaN(0,0).`

`sumaN(N,S+1) :- suma(N-1,S).`

Cómputo para  $n = 5$

resolución del objetivo `sumaN(5,X)` mediante un cálculo lógico.

Respuesta  $X = 15$

# Ejemplo: quicksort (Imperativo)

```
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 := l + 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
```



# Ejemplo: quicksort (Declarativo)

Haskell

```
qsort [] = []
```

```
qsort (x:xs) = (qsort menores) ++ [x] ++ (qsort mayores)
```

```
    where menores = [y | y <- xs, y < x]
```

```
          mayores = [y | y <- xs, y > x]
```

Prolog

```
qsort([], []).
```

```
qsort([X|Xs], S) :- menores(X,Xs,L1), mayores(X,Xs,L2),
```

```
    qsort(L1,S1), qsort(L2,S2), append(S1,[X|S2],S)
```

# PRIMERA PARTE

## CURSO PD

PROGRAMACIÓN FUNCIONAL

Haskell

# Características de Haskell (I)

- **Programas concisos**
  - Lenguaje de alto nivel
  - Pocas palabras clave
  - Usos de indentación para evitar símbolos auxiliares
- **Sistema de tipos muy potente**
  - Inferencia de tipos. Evita errores en ejecución
  - Polimorfismo
  - Sobrecarga de símbolos

# Características de Haskell (II)

- Listas intensionales
  - Listas definidas especificando la propiedad característica de sus elementos
- Funciones recursivas
  - No hay bucles
  - Recursión fácil de definir usando
    - Ajuste de patrones
    - Guardas

# Características de Haskell (III)

- **Funciones de orden superior**
  - Las funciones pueden aplicarse a funciones
  - Las funciones pueden dar como resultado funciones
- **Funciones de efecto total**
  - Previene de los efectos colaterales
  - Mecanismos para no comprometer la pureza del lenguaje
    - Mónadas
    - Functores

# Características de Haskell (IV)

- **Funciones genéricas**
  - Librerías de funciones que pueden aplicarse a muchas estructuras
  - Se pueden definir nuevas estructuras y funciones genéricas sobre ellas
- **Evaluación perezosa**
  - Los cálculos se realizan solo cuando son necesarios
  - Permite terminación y estructuras infinitas

# Características de Haskell (V)

- Razonamiento ecuacional
  - Permite transformación de programas
  - Permite probar propiedades de programas
  - Permite inferir programas a partir de la especificación

# Enlaces documentación Haskell

- [www.haskell.org](http://www.haskell.org) (o googlear 'haskell')
- [www.haskell.org/haskellwiki/Introduction](http://www.haskell.org/haskellwiki/Introduction)
- Descarga del sistema: [www.haskell.org/platform/](http://www.haskell.org/platform/)
- Haskell wiki book: [en.wikibooks.org/wiki/Haskell](http://en.wikibooks.org/wiki/Haskell)
- A Gentle Introduction to Haskell (version 98)  
[www.haskell.org/tutorial/index.html](http://www.haskell.org/tutorial/index.html)
- Haskell report 2010 (definición oficial de Haskell)  
[http://www.haskell.org/haskellwiki/  
Language\\_and\\_library\\_specification](http://www.haskell.org/haskellwiki/Language_and_library_specification)