

# Programación Funcional

  
**Curso 2011-2012**  
**1<sup>er</sup> cuatrimestre**

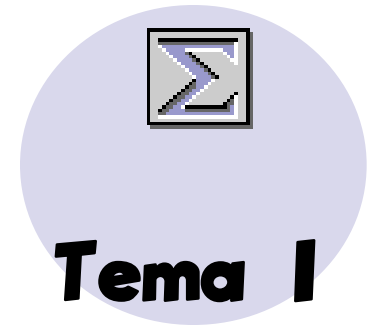
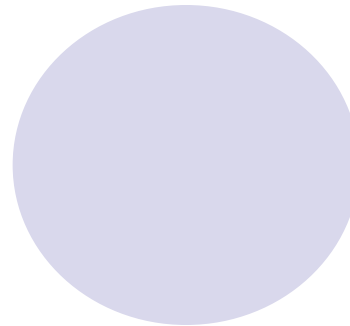
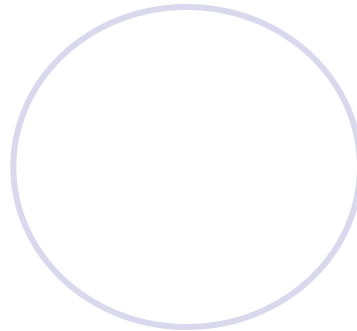
**Paqui Lucio**

**(Despacho 228-FISS)**

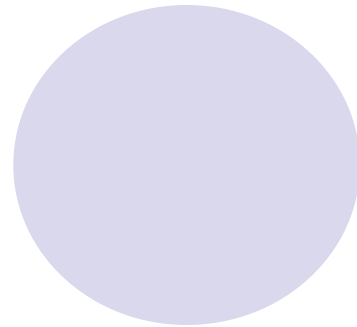
  
**Horario de tutorías: Lunes: 17:30 -18:30**

**Miércoles: 11:00 -13:00**

**Jueves: 15:30 -18:30**



**Tema I**



**Programación Funcional  
y esta Asignatura**





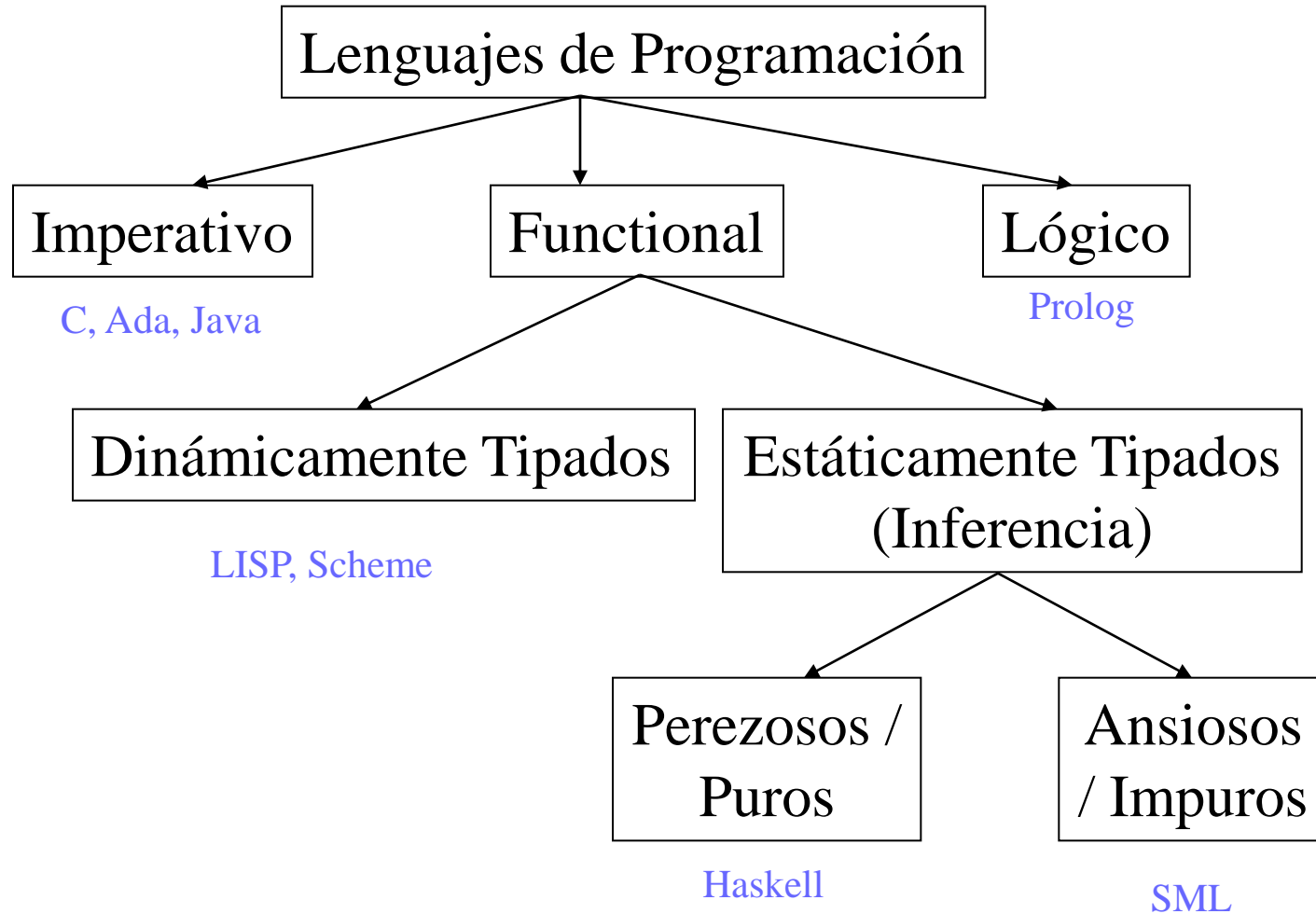
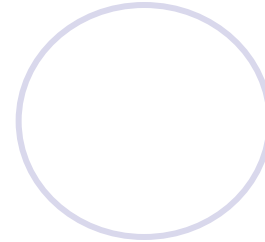
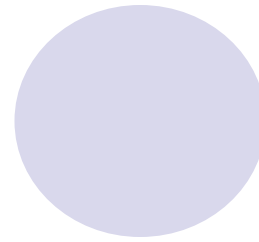
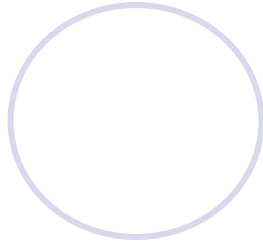
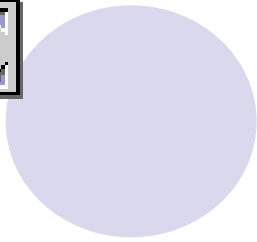
# Programación Funcional

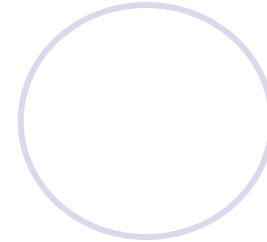
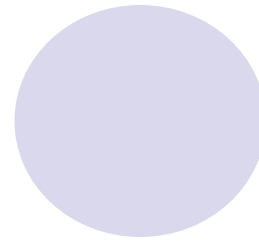
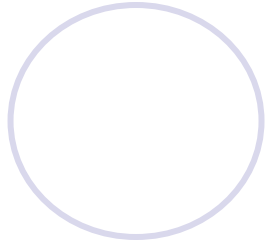
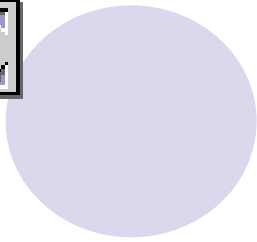


- Paradigma de programación declarativa que trata la computación como la evaluación de funciones matemáticas y evita los estados y datos mutables.
- Énfasis en la aplicación de funciones, en contraste con el estilo de programación imperativo que pone el énfasis en los cambios de estado.

Paradigma	Mecanismo
imperativo	ejecución de comandos
funcional (declarativo)	evaluación de expresiones
lógico (declarativo)	demostración/refutación
etc.	etc.

- Son lenguajes funcionales APL, Erlang, Haskell, ML, Oz, Scheme, ... Algunos muy utilizados en aplicaciones comerciales.





La programación funcional se caracteriza por:

- programar consiste en *definir funciones* por aplicación y composición de otras más simples

```
doble x = x + x
```

```
cuadruple = doble . doble
```

- el método de computación consiste en *evaluar expresiones* que resultan de aplicar funciones (a argumentos) hasta obtener su *valor*

```
cuadruple (doble 2)           (expresión)
```

```
⇒ (doble . doble) (2 + 2)
```

```
⇒ doble (doble 4)
```

```
⇒ doble (4 + 4)
```

```
⇒ (4 + 4) + (4 + 4)
```

```
⇒ 16                           (valor)
```



# Ventajas de la Programación Funcional



- Programas más cortos (de 2 a 10 veces), claros y concisos:

```
quicksort [ ] = [ ]
quicksort (prim:resto)
  = quicksort [y | y<-resto, y<prim] ++
    [prim] ++
    quicksort [y | y<-resto, y>=prim]
```

- Lenguajes (e.g. Haskell) y sistemas (e.g. Hugs) más fáciles para iniciarse y experimentar
- Menos errores, mayor fiabilidad y mejor mantenimiento
  - *Incremento de la productividad del programador (entre 9% y 25% según Ericsson)*
  - *Todo programa (significativo) acabado se modifica muchas veces*



# Principales características de los lenguajes funcionales modernos



- **tipado fuerte** → detección de errores en compilación  
`quicksort (3, 1, 2)` produce un error de tipos  
`quicksort [3, 1, 2]` obtiene `[1, 2, 3]`
- **polimorfismo** → aumenta la reusabilidad  
`quicksort` puede usarse con listas de elementos de cualquier tipo (que cuente con una operación `<`)
- **evaluación perezosa** → modularidad, facilidad de escritura  
`pares = [2*x | x <- [1..]]`  
`take 5 pares`



## Principales características ... (cont.)



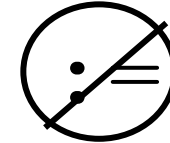
- funciones de orden superior → alto nivel de abstracción
  - *funciones como “ciudadanos de primera clase”*
  - *aplicación/composición de funciones → modularidad*  
cuadruple = doble . doble
- gestión de memoria → libera de carga al programador  
*inicialización, almacenamiento y liberación automática*
- Un programa es una serie de definiciones y una ejecución es la evaluación de una expresión que usa las entidades definidas en el programa.



## Principales características ... (cont.)



- Los programas funcionales se basan en dar definiciones usando el símbolo de igualdad (=), que nada tiene que ver con la asignación.



- En programación funcional, el concepto de posición de memoria a la que se le asigna un valor no existe

- Si en un programa aparece

`x = <expresión>`

`x es, por definición, igual a <expresión>`

- Por tanto, en ese mismo programa no puede aparecer otra vez

`x = ...`

de lo contrario no sería correcto.



# El Lenguaje Funcional Haskell

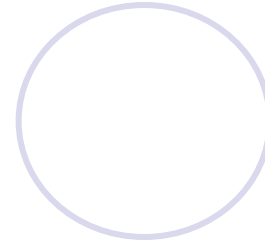
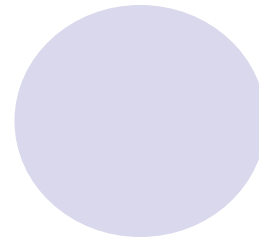
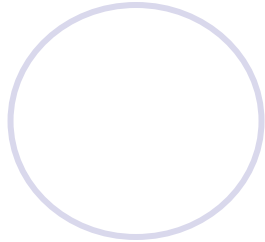
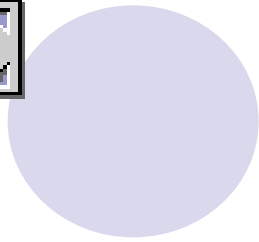


- Haskell es el lenguaje funcional más estándar (hoy en día)
- Su nombre se debe a Haskell Brooks Curry (1900-1982), estadounidense cuyo trabajo en lógica matemática ha servido de amplio fundamento a la programación funcional

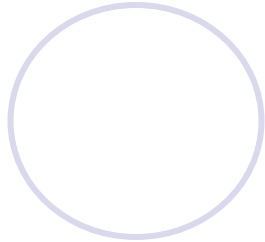
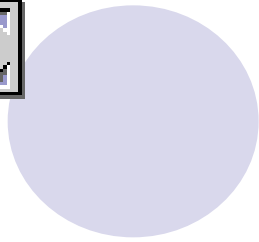


Trabajó en el primer ordenador electrónico llamado **ENIAC** (Electronic Numerical Integrator and Computer) después de la Segunda Guerra Mundial.

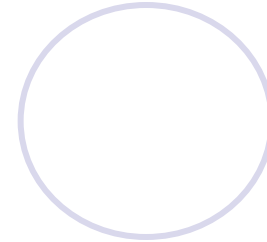
En 1986 la Mitre Corporation creó el Curry Chip, un innovador elemento hardware basado en el concepto matemático de combinador que Curry definió en los 50's, exactamente del mismo modo que están basadas las implementaciones de lenguajes funcionales (p.e. Miranda y Haskell).



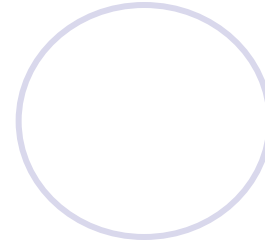
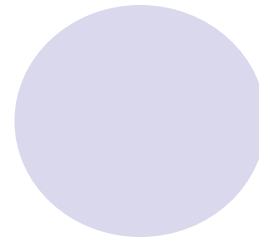
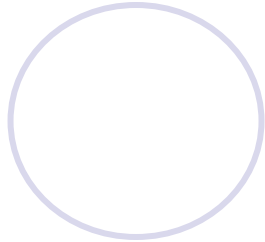
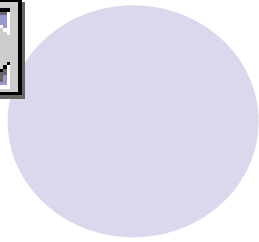
- Haskell está basado (semántica e implementación) en el  $\lambda$ -cálculo
- Haskell se usa en muchas universidades como primer lenguaje de programación
- Haskell es ampliamente usado en la industria para prototipado rápido de programas y en diseño y verificación de hardware
- Haskell posee todas las características antes comentadas y algunas otras como
  - clases de tipos,
  - facilidades de módulos,
  - I/O puramente funcional,etc



## El Sistema Hugs



- Hugs (Haskell User's Gofer System) es la implementación de Haskell que utilizaremos en el laboratorio
  - Gofer significa Good for equational reasoning
- Programar en Haskell consiste en definir funciones en un “*script*”:
  - “*script*” = guión, manuscrito
  - fichero que contiene definiciones, declaraciones y comentarios  
( = programa)
  - <nombre>.hs (haskell script)
- Almacenando (o cargando) el “*script*” en Hugs, las funciones definidas pueden ser usadas en la sesión
- Además Hugs conoce (por defecto) una amplia colección de funciones y operadores, que están contenidas en Prelude.hs



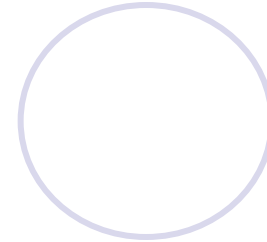
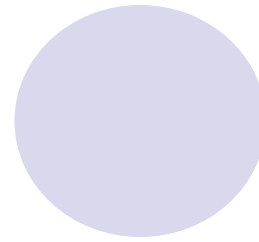
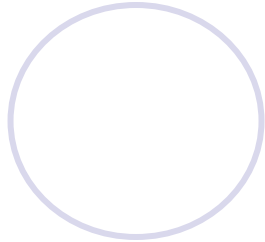
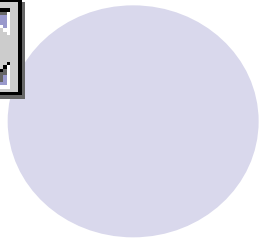
- Ejecutar en Hugs es evaluar una expresión en base a los programas cargados en la sesión:

```
Hugs sesion for: ...../Prelude.hs
Type :? for help
Prelude>
```

```
Prelude> 2+3*4
14
Prelude>
```

```
Prelude> (2+3) *4
20
Prelude>
```

```
Prelude> length [1,2,3,4]
4
Prelude> take 3 [1,2,3,4]
[1,2,3]
```



- Si el script `prueba.hs` consiste en las definiciones

```
doble x = x + x
```

```
pares = [doble x | x ← [1..]]
```

el efecto de `%hugs prueba.hs` es

```
Hugs sesion for:
..... ./Prelude.hs
prueba.hs
Type :? for help
Main>
```

ahora se puede usar como calculadora (con las funciones pre-definidas más las definidas por el usuario):

```
Main> take 5 pares
[2,4,6,8,10]
```



## Expresiones indefinidas



- No toda expresión tiene un valor (aunque sea correcta y bien tipada):

```
Prelude> 1/0
Program error: {primDivDouble 1.0 0.0}
Prelude> sqrt (-1)
Program error: {primSqrtDouble (-1.0)}
```

- Si definimos

```
reset x = 0
infinito = infinito + 1
```

entonces:

- `infinito :: Integer` no tiene valor (por *evaluación infinita*)
  - `reset infinito` y `reset 1/0` se evalúan perezosamente a 0
- Denotaremos por  $\perp$  a cualquier expresión bien tipada pero indefinida bien sea por *evaluación infinita* o por *error*



## Contenido de un “script”



- Definiciones:

```
doble x = x + x
```

es un definición (escrita en Haskell) de la función `doble`

- Declaraciones:

```
doble :: Int -> Int
```

es un declaración de tipo en Haskell

El sistema puede inferir el tipo de `doble` de la definición de arriba.

- Comentarios:

```
-- doble aplicada a un entero devuelve  
-- ese entero multiplicado por 2
```

es un comentario que el intérprete ignora



## La evolución del programador de Haskell

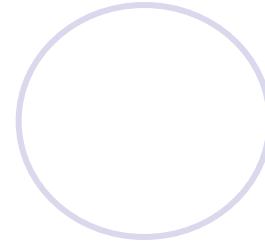
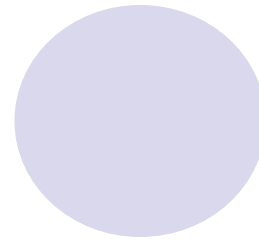
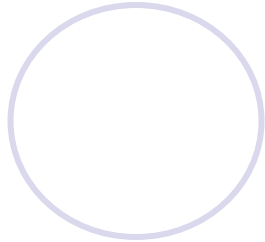
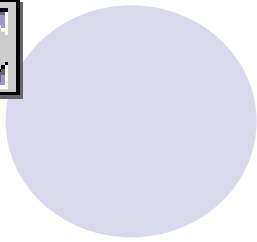


- Haskellero iterativo (*Programador habitual de Pascal/Ada*)

```
fac n = result (for init next done)
      where init = (0,1)
            next (i,m) = (i+1, m * (i+1))
            done (i,_) = i==n
            result (_,m) = m
for i n d = until d n i
```

- Haskellero iterativo (*Programador habitual de C/APL*)

```
fac n = snd(until ((>n) . fst) (\(i,m) -> (i+1, i*m)) (1,1))
```



- Haskellero junior

```
fac 0 = 1
```

```
fac (n+1) = (n+1) * fac n
```

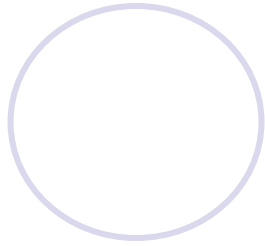
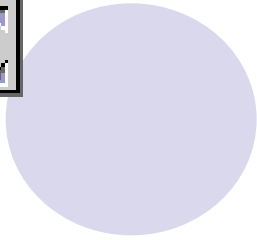
- Haskellero senior

```
fac n = foldr (*) 1 [1..n]
```

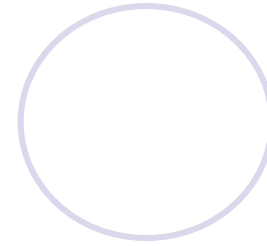
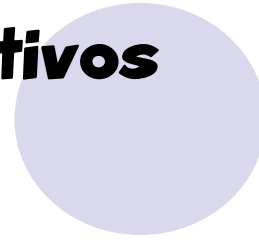
- Haskellero fundamentalista

```
fac = product . enumFromTo 1
```

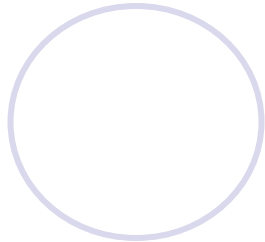
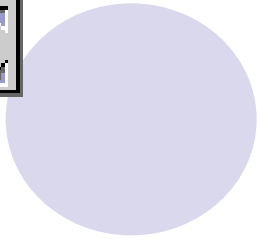
Véase: <http://www.willamette.edu/~fruehr/haskell/evolution.html>



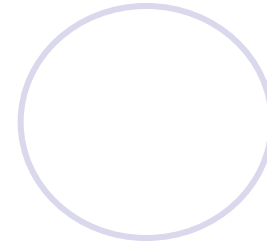
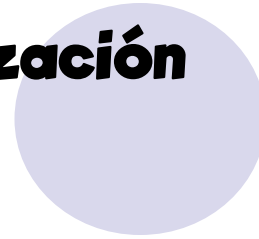
## Objetivos



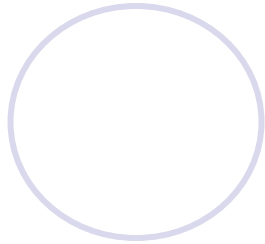
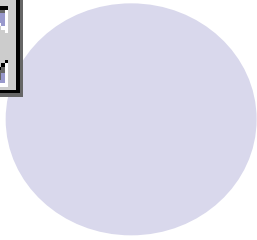
- Estudiar las principales características de los lenguajes funcionales modernos
- Adquirir la destreza necesaria para realizar programas propios del paradigma funcional en tiempo y forma razonable
  - Lenguaje Haskell
  - Software Libre: Hugs (intérprete de Haskell, <http://www.haskell.org>)



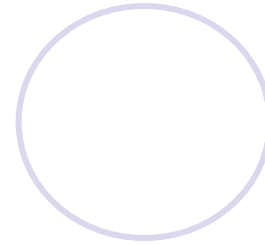
## Organización



- Créditos: 4 Teóricos + 2 Prácticos
- Clases
  - explicaciones
  - ejercicios
- Laboratorios
  - trabajo personal dirigido
  - Los primeros para conocer el sistema y ejercitar lo aprendido en clase
  - Los últimos para realizar la *práctica*



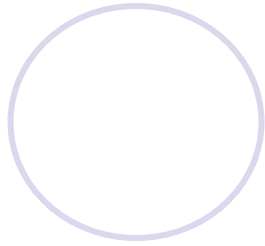
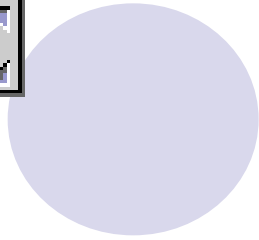
## Documentación



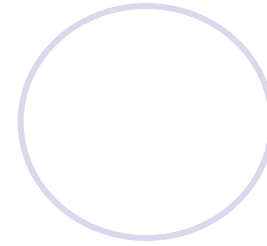
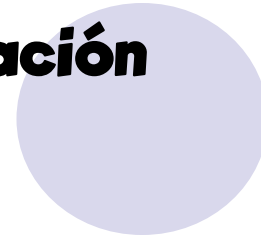
- Página Web de la asignatura: <http://www.sc.ehu.es/jiwlucap/PF.html>

donde irán estando accesibles:

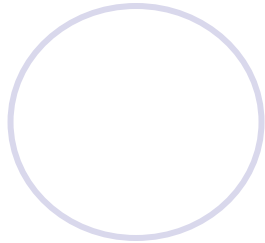
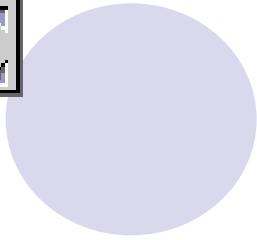
- Las transparencias que se explican en clase
- Las listas de ejercicios que se utilizan en clase
- Los guiones de las sesiones de laboratorio
- Las soluciones de los laboratorios (a final del cuatrimestre)
- El enunciado y ficheros auxiliares de la práctica



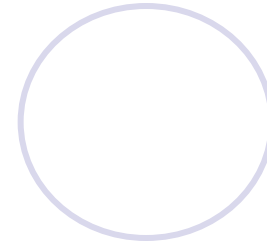
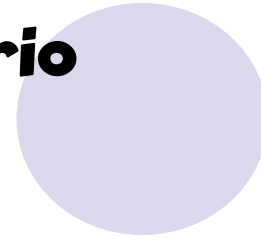
## Evaluación



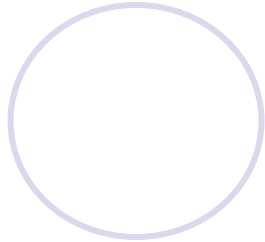
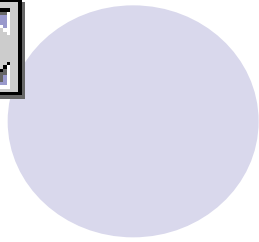
- Para aprobar la asignatura en la convocatoria de Febrero hay que
  - superar el examen final y
  - entregar un trabajo práctico que se podrá realizar durante aproximadamente la mitad de los laboratorios lectivos.
- La calificación final en Febrero será:
  - examen (sobre 7 puntos) + práctica (sobre 3 puntos)
- En Junio, sólo habrá un examen sobre 10 puntos.



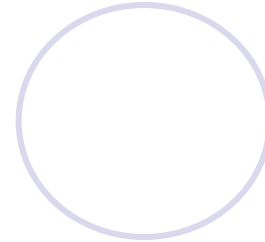
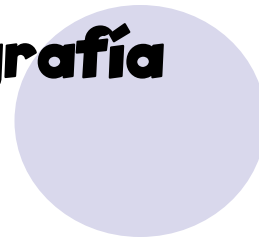
# Temario



- 1.- Programación Funcional y esta Asignatura
- 2.- Tipos y Clases
- 3.- Funciones y Operadores
- 4.- Clases Básicas, Instancias y Tipos Básicos
- 5.- Tuplas
- 6.- Definición de Funciones
- 7.- Listas y Strings
- 8.- Tipos Algebraicos
- 9.- Programando con Acciones: I/O
- 10.- Estructuras Infinitas y Cíclicas



# Bibliografía



## Básica

- *Introducción a la Programación Funcional con Haskell*  
R. Bird  
Prentice Hall, 1998 (v.o.), 2000 (traducción al español)
- *Haskell: The Craft of Functional Programming*  
S. Thompson  
Addison-Wesley, 1999

## Página Web de Haskell:

- <http://www.haskell.org>