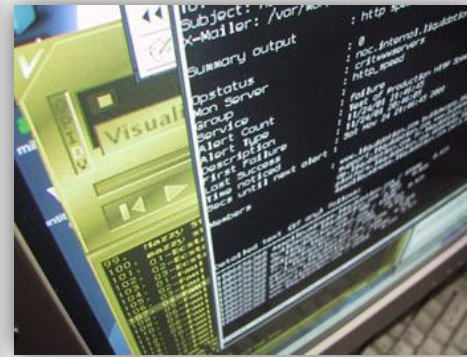




# El lenguaje Python

1. Introducción
2. Historia
3. Filosofía
4. Características
  - a. Tipos de datos
  - b. Expresiones y control de flujo
  - c. Funciones
5. Python VS Haskell
  - a. Listas por compresión
  - b. Funciones sobre listas
  - c. Funciones  $\lambda$  (lambda)

6. Estado del lenguaje
  - a. Uso
  - b. Aplicaciones que usan python
7. Ejemplos
8. Bibliografía
9. Y para terminar
  - a. Experiencia
  - b. Conclusiones



# Bloque 1

# Introducción

1. **Introducción**
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Podemos destacar

Es un lenguaje de programación de alto nivel y de propósito general. El uso de sangrías como delimitadores de bloques no es usual.

Python permite múltiples paradigmas de programación y contiene un completo sistema de tipos dinámicos y gestión de memoria.

Tiene la misma filosofía que Linux. Y es gestionado por la organización Python Software Foundation.

1. Introducción
2. **Historia**
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

**¿Cuándo nació? ¿Qué ha pasado desde entonces?**

Fue concebido a finales de los años 80

Su antecesor fue el lenguaje ABC

Python se hizo público el 16 de octubre de 2000

Actualmente la versión que tenemos disponible es la de Python 3.0, publicada en diciembre de 2008



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

**Tiene una filosofía multi-paradigma**

**La programación orientada a objetos y la estructurada está completamente soportada**

**Incorpora ciertas funcionalidades que permiten la programación FUNCIONAL y orientada a aspectos**

**Python es de tipificado dinámico y resolución dinámica de nombres**

1. Introducción
2. Historia
3. **Filosofía**
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

**Admite parcialmente la programación funcional al estilo LISP**

**Existen varios módulos que implementan herramientas funcionales prestadas de Haskell y Standard ML**

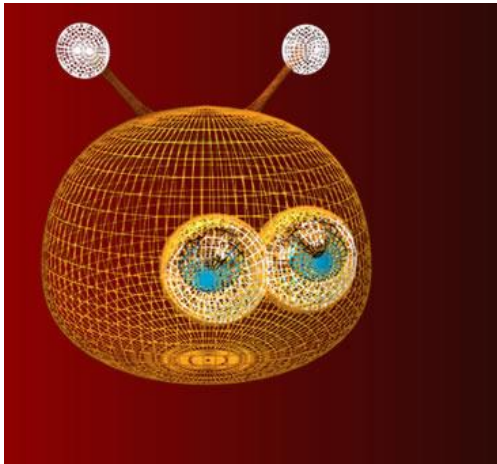
**Rechaza las sintaxis exuberantes**



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Zen de Python:

1. Bonito es mejor que feo
2. Simple es mejor que complejo
3. Plano es mejor que anidado
4. La legibilidad cuenta
5. Debería haber una manera (y preferiblemente sólo una) obvia de hacerlo
6. Los namespaces son buena idea  
¡Hagamos más!



```
0100000000010100011011000000100101100011
110001011101000100011111111110100000100
00101001011000011010111011010110010001
0110110000010101100100010000111000100111
010011001011010011011010011110111011110
00011010011010011010011101000011010
1001001101001001001010001110
110001001101001001000101111
0101010011010010010001100011000
111001100110011001100110011001100
0010000011111001100110101110110
0001101000100011101001110001101000011010
0100100110111101011101110000001010001110
11000100100010101100100111011101000101111
01010100111001101010111000101010100011000
1110011000001101111110101001111110001100
0100000111111101010010010011010101110110
```

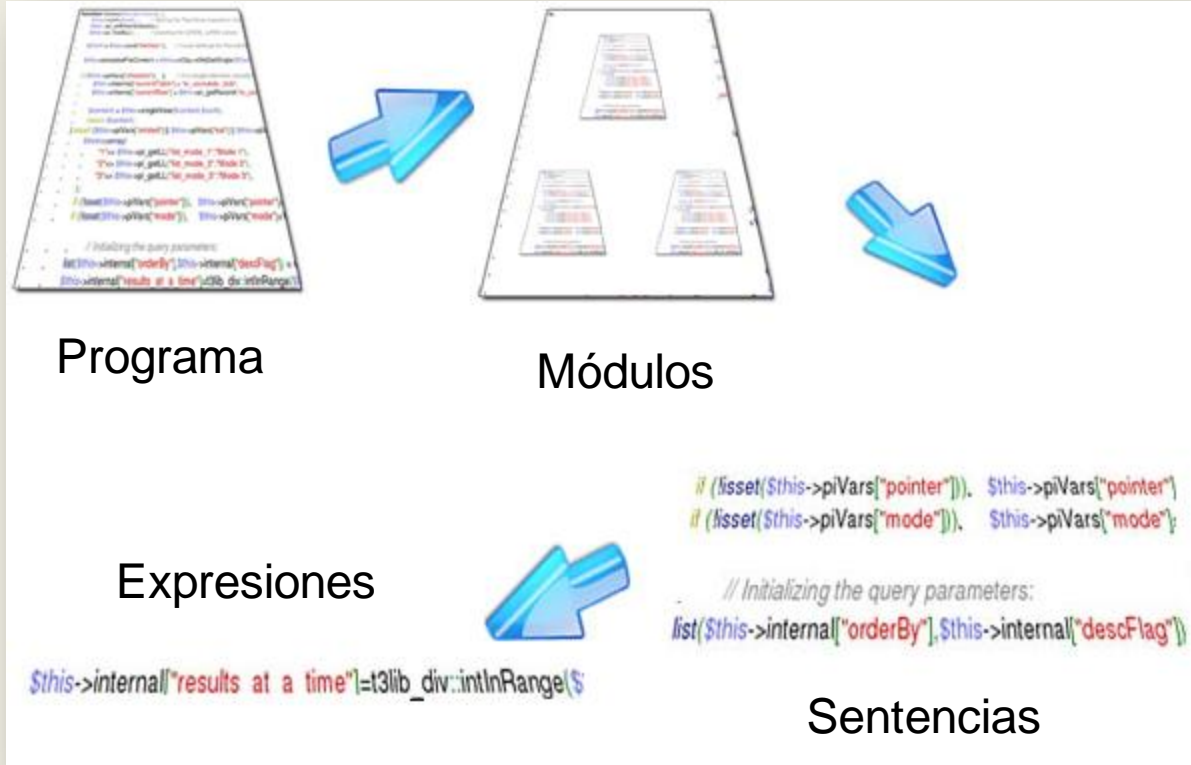


## Bloque 2

# Características

1. Introducción
2. Historia
3. Filosofía
4. **Características**
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

**Un programa en Python puede ser descompuesto en módulos, sentencias, expresiones y objetos**



1. Introducción
2. Historia
3. Filosofía
4. Características
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Objetivos y tipos de datos

En Python todo esta representado mediante objetos o relaciones entre objetos

Cada objeto tiene una identidad, un tipo y un valor

Los objetos nunca son explícitamente destruidos

Python usa el denominado “duck typing” y tiene objetos tipificados y variables no tipificadas

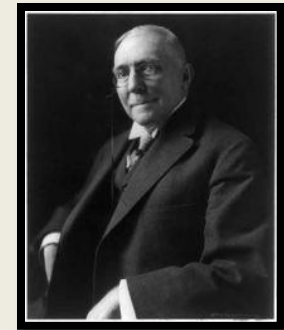
*¿duck typing?*

1. Introducción
2. Historia
3. Filosofía
4. **Características**
  - a. **Objetos y tipos de datos**
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## ¿duck typing?

Si camina como un pato, nada como un pato y hace “quack” como un pato... Yo diría que es un pato

**James Whitcomb Riley**



Para esta función el objeto que recibe es un pato

1. Introducción
2. Historia
3. Filosofía
4. Características
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## ¿duck typing?

```

class Duck:
    def quack(self): print "Quaaaaaack!"
    def feathers(self): print "The duck has white and gray feathers."

class Person:
    def quack(self): print "The person imitates a duck."
    def feathers(self): print "The person takes a feather from the \
ground and shows it."

def in_the_forest(duck):
    duck.quack()
    duck.feathers()

def game():
    donald = Duck()
    john = Person()
    in_the_forest(donald)
    in_the_forest(john)
  
```

1. Introducción
2. Historia
3. Filosofía
4. **Características**
  - a. **Objetos y tipos de datos**
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Python proporciona una serie de tipos predefinidos

Tipo	Clase	Descripción
<i>str</i>	String	Secuencia inmutable de caracteres
<i>unicode</i>	String	Versión Unicode de <i>str</i>
<i>list</i>	Sequence	Secuencia mutable de objetos de tipo arbitrario
<i>tuple</i>	Sequence	Secuencia inmutable de objetos de tipo arbitrario
<i>set</i>	Set	Conjunto mutable de objetos sin orden de tipo arbitrario
<i>dict</i>	Mapping	Grupo mutable de pares clave-valor
<i>int</i>	numbers.Integral	Entero de magnitud no definida, sólo limitada por la memoria disponible
<i>float</i>	numbers.Real	Números reales en punto flotante de doble precisión. Rango de valores dependiente de la máquina
<i>complex</i>	numbers.Complex	Números complejos representados mediante un par de números de punto flotante.
<i>bool</i>	numbers.Integral	Valores de verdad

# Características

1. Introducción
2. Historia
3. Filosofía
4. **Características**
  - a. Objetos y tipos de datos
  - b. **Sentencias**
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Sentencia	Rol	Ejemplo
Asignación	Creación de referencias	<code>a, b, c = 'good', 'bad', 'ugly'</code>
Llamada	Ejecución de funciones	<code>log.write("spam, ham\n")</code>
<i>print</i>	Impresión de objetos	<code>print 'The Killer', joke</code>
<i>if/elif/else</i>	Selectiva	<code>if "python" in text: print text</code>
<i>for/else</i>	Iteración sobre una secuencia	<code>for x in mylist: print x</code>
<i>while/else</i>	Bucle	<code>while X &gt; Y: print 'hello'</code>
<i>break, continue</i>	Salto en bucle	<code>while True: if not line: break</code>
<i>pass</i>	Sentencia nula	<code>while True: pass</code>
<i>assert</i>	Aserción	<code>assert x &gt; y</code>
<i>try/except/finally</i>	Captura de excepciones	<code>try: action() except: print 'action error'</code>
<i>raise</i>	Lanzar excepciones	<code>raise endSearch, location</code>
<i>import, from</i>	Acceso a módulos	<code>import sys from sys import stdin</code>
<i>def, return, yield</i>	Definición de funciones	<code>def f(a, b, c=1, *d): return a+b+c+d[0] def gen(n): for i in n, yield i*2</code>
<i>class</i>	Definición de clases	<code>class subclass(Superclass): staticData = []</code>
<i>global</i>	Declaraciones globales	<code>def function(): global x, y x = 'new'</code>
<i>del</i>	Eliminación de referencias	<code>del data[k] del data[i:j] del obj.attr del variable</code>
<i>with/as</i>	Creación de contextos	<code>with open('data') as myfile: process(myfile)</code>



1. Introducción
2. Historia
3. Filosofía
4. Características
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Son la estructura más básica que proporciona Python

```
def <identificador>(arg1, arg2,..., argN):
    ["Documentación"]
    <sentencias>
```

```
def fib(n): # return Fibonacci series up to n
    """Return a list containing the Fibonacci series up to n."""
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

```
>>> fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

1. Introducción
2. Historia
3. Filosofía
4. Características
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. Módulos
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Si se cierra el editor de Python y se vuelve a abrir, las definiciones que existían se pierden

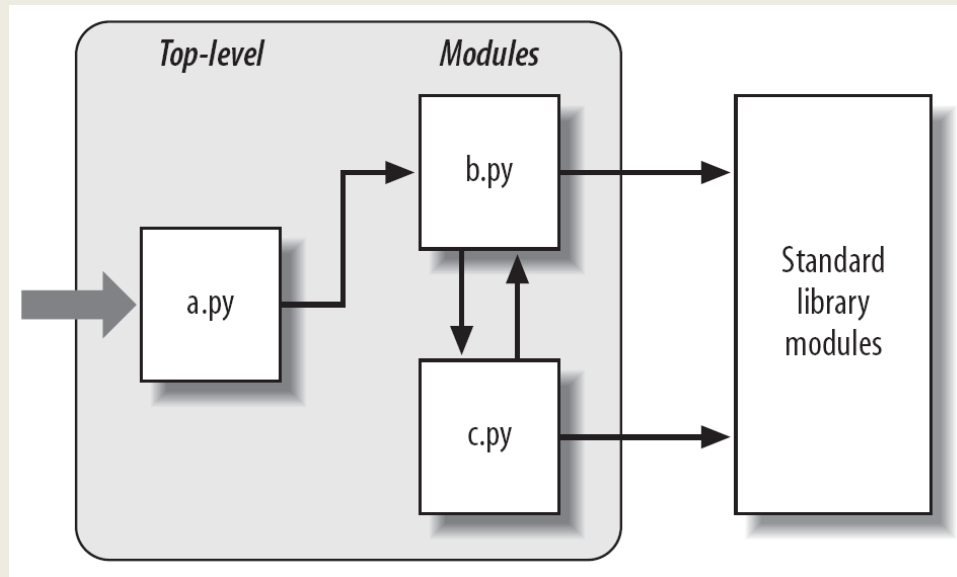
**¿qué hacemos?**

Debemos utilizar un editor, para poder definir las y guardarlas (\*.py)

Las definiciones de un módulo se pueden importar hacia otros módulos o hacia el módulo principal

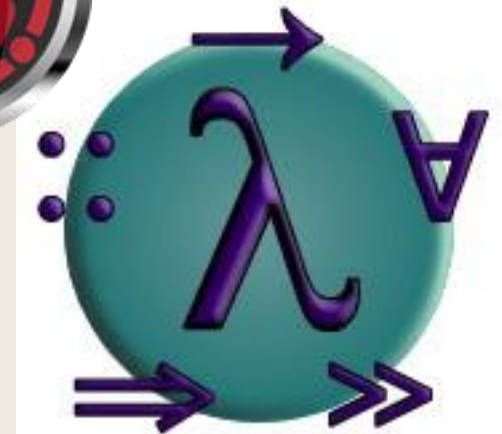
# Características

1. Introducción
2. Historia
3. Filosofía
4. **Características**
  - a. Objetos y tipos de datos
  - b. Sentencias
  - c. Funciones
  - d. **Módulos**
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

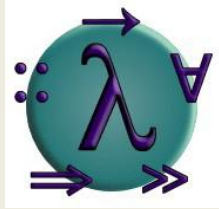


Python usa los espacios en blanco como separadores de bloques

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



Notación similar a la de conjuntos por comprensión

$[ \textit{expresión} \mid \textit{cualificador}_1, \textit{cualificador}_2, \dots, \textit{cualificador}_n ]$

Cada cualificador puede ser:

- Un generador,
- Una expresión booleana, o
- Una definición local

```
Prelude> [2*x | x <- [0..5], even x]
[0,4,8]
Prelude>
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



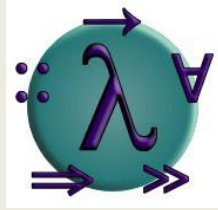
## La sintaxis difiere ligeramente de la de Haskell

```
[ expresión  for expresión in secuencia1
  if condición2
  for expresión in secuencia2
  if condición2
  ...
  for expresión in secuencian
  if condiciónn ]
```

El ejemplo anterior sería ahora de la forma

```
IDLE 1.2.1
>>> [ 2*x for x in range(6) if x % 2 == 0 ]
[0, 4, 8]
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



map  
filter  
reduce

```
map      :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = f x : map f xs
```

Aplica una función a todos los elementos de la lista, devolviendo una lista con los resultados

```
Char> map ord "pepe"
[112,101,112,101]
Char> map (^ 2) [1,2,3]
[1,4,9]
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



map  
filter  
reduce



$map(f, iter_1, iter_2, \dots, iter_n)$

$f(iter_1[0], iter_2[0], \dots, iter_n[0]), f(iter_1[1], iter_2[1], \dots, iter_n[1]), \dots$

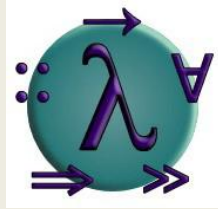
El mismo ejemplo en versiones distintas 2.5.1 y la 3.0

```
>>> map(ord, "hola")
[104, 111, 108, 97]
```

```
>>> map(ord, "hola")
<map object at 0x00F75250>
```



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



map  
filter  
reduce

```
filter      :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (x : xs)
  | p x = x : filter p xs
  | otherwise filter p xs
```

Esta función nos permite seleccionar los elementos de una lista que cumplen cierta propiedad

```
Prelude> filter (> 'g') "me gustan mucho las listas"
"mustmuholslists"
Prelude> filter even [1..20]
[2,4,6,8,10,12,14,16,18,20]
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



map  
filter  
reduce



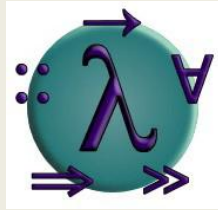
`filter(predicado, iterador)`

El predicado es la función que determina el cumplimiento de la condición y se devuelve un iterador sobre la secuencia de elementos que cumplen la condición

```
>>> filter((lambda x : x > 'g'), 'me gustan mucho las listas')
'mustmuholslists'
>>> def is_even(x) :
        return x % 2 == 0

>>> filter(is_even, range(1,20))
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



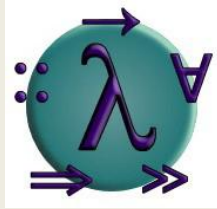
map  
filter  
reduce

```
foldl1 :: (a -> a -> a) -> [a] -> a
foldl1 f (x : xs) = foldl f x xs
```

Para realizar operaciones de forma acumulativa

```
Prelude> foldl1 max [1,2,3,4]
4
Prelude> foldl1 (+) [1..10]
55
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



El `foldl1` realiza un plegado de las listas de izquierda a derecha

```

foldl1 (+) [1..10]
=>
foldl1 (+) 1 [2..10]
=>
foldl1 (+) (+ 1 2) [3..10]
=>
foldl1 (+) (+ (+ 1 2) 3) [4..10]
=>
foldl1 (+) (+ (+ (+ 1 2) 3) 4) [5..10]
=>
...
  
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...



map  
filter  
reduce



En Python si disponemos de la función reduce como tal, se encuentra dentro del módulo *functools*

```
functools.reduce(funcion, iterador, [valor_inicial])
```

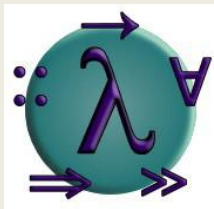
```
>>> def suma(x, y) :
        return x + y
>>> reduce(suma, [1,2,3,4])
10
>>> reduce(max, [1,2,3,4])
4
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
  - a. Listas por comprensión
  - b. Funciones sobre listas
  - c. Operador Lambda
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

A veces necesitamos pequeñas funciones que actúen como predicados o que combinen elementos de alguna manera

Estas funciones se denominan anónimas o lambda

## ¿Cómo se definen?



$$\lambda \text{ arg}_1 \text{ arg}_2 \dots \text{ arg}_n \rightarrow \text{expresión}$$
`lambda arg1 arg2 ... argn : expresión`

`\ x y z -> x + y + z`
`lambda x, y, z : x + y + z`

```
alCubo :: Int -> Int
alCubo = \ x -> x ^ 3
Main> alCubo 4
64
```

```
>>> (lambda x : x**3) (4)
64
```

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
  - a. Uso
  - b. Aplicaciones que usan Python
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Se usa frecuentemente para aplicaciones web, por ej. Servidor Apache



Servidores de aplicaciones como



Se ha usado mucho dentro de la industria de la seguridad de información

Python es un componente estándar



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
  - a. Uso
  - b. Aplicaciones que usan Python
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Entre otros famosos usuarios de Python podemos encontrar



Varias organizaciones también hacen uso de este lenguaje





1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
  - a. **Uso**
  - b. Aplicaciones que usan Python
7. Ejemplos
8. Bibliografía
9. Y para terminar...



También lo podemos encontrar dentro de la educación



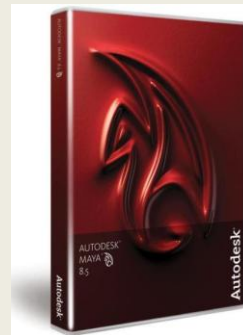
1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
  - a. Uso
  - b. Aplicaciones que usan Python
7. Ejemplos
8. Bibliografía
9. Y para terminar...

Se incluye en numerosos productos software como lenguaje Script



Análisis de elementos finitos

Animación 3D



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
  - a. Uso
  - b. Aplicaciones que usan Python
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Tratamiento de imágenes



**GIMP**



**Inkscape**



**Scribus**

## Sistemas de información geográfica



1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A	Black	White	Black	White
B	White	Black	White	Black
C	Black	White	Black	White
D	White	Black	White	Black

```
def reinas(n) :
    return [s for s in itertools.permutations(range(1,n+1)) if buena(s)]
```

[1, 2, 3, 4]

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A	+			
B		+		
C			+	
D				+

[1, 2, 3, 4]

noAtaca 1 [2, 3, 4]

**False**

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A		+		
B				+
C	+			
D			+	

[2, 4, 1, 3]

noAtaca 2 [4, 1, 3]  
True

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A		+		
B				+
C	+			
D			+	

[2, 4, 1, 3]

noAtaca 4 [1, 3]  
True

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A		+		
B				+
C	+			
D			+	

[2, 4, 1, 3]

noAtaca 1 [3]  
True



# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Las 4 reinas

	1	2	3	4
A		+		
B				+
C	+			
D			+	

4!

[2, 4, 1, 3]

noAtaca 3 []  
True

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## ¿y las 8 reinas?

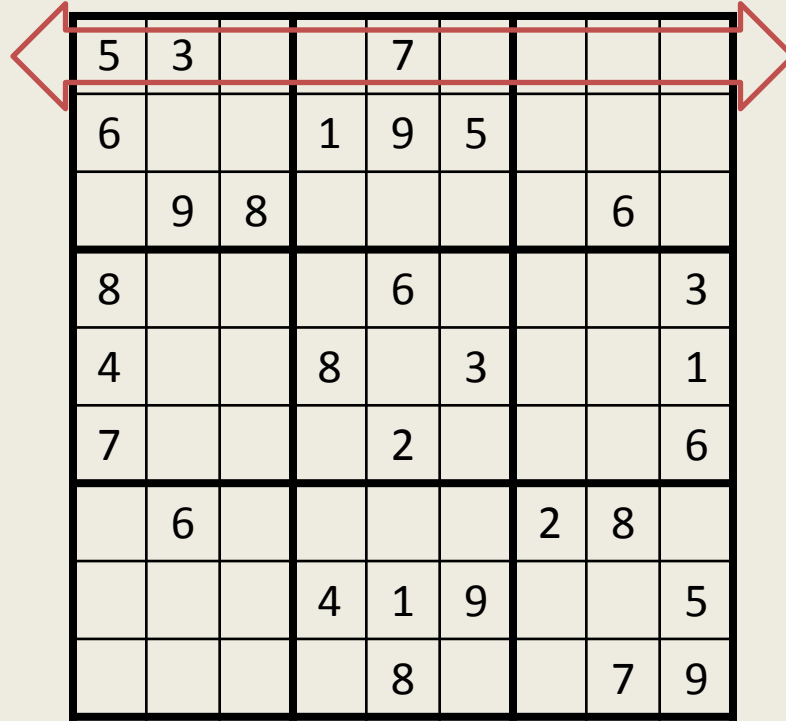
	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

**8! = 40.320**

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Sudoku



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

[5, 3, 7]

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

[5, 3, 7, 8]

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

[5, 3, 7, 8, 6, 9]

# Ejemplos

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. **Ejemplos**
8. Bibliografía
9. Y para terminar...

## Sudoku

5	3	1		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

[5, 3, 7, 8, 6, 9]  
 '1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~'

# Bibliografía

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. **Bibliografía**
9. Y para terminar...

1. Mark Lutz. Learning Python. O'Reilly, 2007
2. Blas C. Ruiz, Francisco Gutierrez, Pablo Guerrero, y José E. Gallardo. Razonando con Haskell. Un curso sobre programación funcional. Thomson, 2004
3. Wikipedia: Python (programming language), actualizado 05/2009 [\[1\]](#)
4. Wikipedia: Python, actualizado 05/2009 [\[2\]](#)
5. Python v3.0.1 documentation, actualizado 05/2009 [\[3\]](#)
6. Charming Python: Functional programming in Python, Part 1, 2, and 3, actualizado 06/2009 [\[4\]](#) [\[5\]](#) [\[6\]](#)

# Concluyendo

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...
  - a. Experiencia
  - b. Conclusiones

1. Es un lenguaje agradable y de fácil aprendizaje
2. Lenguaje en auge y bastante extendido
3. Al ser de código abierto existe mucha documentación
4. El intérprete deja mucho que desear
5. Es fácil portar programas de otros lenguajes a Python
6. La última versión no es completamente compatible con las anteriores



# Concluyendo

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...
  - a. Experiencia
  - b. Conclusiones

Python es rápido

Extensible

Portable

Extensa librería estándar

Tipificado dinámico

Código abierto

Python no es un lenguaje funcional puro

Compatibilidad entre versiones

Recursión limitada

No tiene comparación de patrones

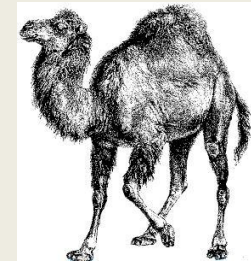
No utiliza evaluación perezosa

# Concluyendo

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

- a. Experiencia
- b. Conclusiones

## ¿Quién es mejor?



Es mejor debido a que es más fácil de aprender y tiene un código más legible

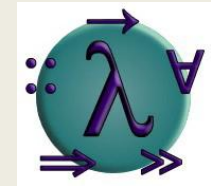


Su código es de 5 a 10 veces más conciso y dispone de tipificado dinámico

# Concluyendo

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...
  - a. Experiencia
  - b. Conclusiones

## ¿Quién es mejor?



Al no disponer de comparación de patrones ni de evaluación perezosa, sin duda alguna para programación funcional, Haskell es mucho mejor que Python

1. Introducción
2. Historia
3. Filosofía
4. Características
5. Python VS Haskell
6. Estado del lenguaje
7. Ejemplos
8. Bibliografía
9. Y para terminar...

## Ruegos y Preguntas



# Autores:

Sergio Paque Martin  
[pakesoy@gmail.com](mailto:pakesoy@gmail.com)

David Abolafia Cañete  
[abolafia@gmail.com](mailto:abolafia@gmail.com)

5º de Ingeniería Informática  
(2008/2009)



Esta obra está bajo una [licencia de Creative Commons](https://creativecommons.org/licenses/by/4.0/).