

# Programación Científica y HPCI

Máster Universitario en Ingeniería Matemática y Computación

## Tema 1

# ¿Cómo estudiar este tema?

1. Introducción y objetivos
2. Clasificación de lenguajes de programación
3. Lenguajes científicos: R, Scala, MATLAB/Octave, Python, C/C++, Fortran
4. Evaluación de rendimiento: profiling
5. Gestión de cambios: git

## Material Complementario

[Paradigmas de programación](#)

[Zine de profiling y trazado de programas](#)

[Tutorial de 30 minutos en vídeo para empezar a usar git](#)

[Practicar conceptos avanzados de git en el navegador](#)

[Referencia rápida de git en español](#)

## Magistrales

### Tema 1. Uso del escritorio de Git



# Clasificación de los lenguajes

- Alto nivel

- Más explícito para el humano
- Imposible de procesar por una máquina

```
If (a > b) then
    a = a - b;
else
    b = b + 1;
```

- Bajo nivel

- Difícil para el humano
- Entendido por una máquina

```
MOVE t1 a
MOVE t2 b
CMP t1 t2
BZ E1
SUB t3 t1 t2
MOVE a t3
E1: MOVE t4 #1
ADD t5 t2 t4
MOVE b t5
```

```
LD #20[.IX] /100
LD #21[.IX] #12[.IX]
CMP #20[.IX] #21[.IX]
BZ E1
SUB #22[.IX] #20[.IX] #21[.IX]
LD /100 #22[.IX]
E1: LD #23[.IX] #1
ADD #24[.IX] #21[.IX] #23[.IX]
LD #21[.IX] #24[.IX]
```

# Clasificación de los lenguajes

## • Compilado



- Realiza todas las comprobaciones antes de la ejecución
- Traduce antes de la ejecución
- Código máquina para arquitectura nativa

- Mas rápida la ejecución
- Menos errores de ejecución
- No requiere del compilador para la ejecución

C++,Ada, Java,  
Python?

## • Interpretado



- Realiza comprobaciones en ejecución
- Traduce en ejecución
- La mayoría genera código para una máquina virtual

- Ciclo de desarrollo más rápido
- Mayor portabilidad
- Se requiere del interprete para la ejecución
- Más errores de ejecución

Python, Java?

# Clasificación de los lenguajes

- Tipificado estático

- Se enlaza el tipo con la variable en tiempo de compilación
- Permite detectar errores de tipos

C++, Ada, Java

```
· lista l=new lista;
```

```
l=3;;error!
```

- Bajo nivel

- Se enlaza el tipo con la variable en tiempo de ejecución
- Se pueden producir errores de tipos en tiempo de compilación

```
· l=list()
```

```
l
```

```
Out[2]: []
```

```
l=3
```

```
Out[4]: 3
```

Python, Lisp



# Paradigmas de programación

- Programación imperativa: serie de instrucciones o sentencias → Pascal, C, C++, Java, Python
- Programación declarativa: describe el problema → Prolog, Lisp, Python
- Programación orientada a objetos → C++, Java, Python
- Programación reflexiva: modifica estructura en tiempo de ejecución → Java, Python, Smalltalk
- Programación concurrente, distribuida ...



- Estructurada
- Procedimientos
- Módulos

- Razonamiento lógico

- Objetos
- Atributos
- Métodos

- Máquina virtual
- Metadatos

# Lenguajes de programación científica

| Logo  | Lenguaje             | Nivel                | Tipificado | Traducción   | Usos principales  |
|---|----------------------|----------------------|------------|--------------|---|
|    | <b>C/C++</b>         | Bajo (C), Alto (C++) | Estático   | Compilado    | Computación de alto rendimiento, librerías matemáticas, computación heterogénea |
| -   | <b>Fortran</b>       | Alto                 | Estático   | Compilado    | Computación de alto rendimiento, librerías matemáticas                          |
|    | <b>Python</b>        | Alto                 | Dinámico   | Interpretado | Ciencia de datos, big data, aprendizaje automático                              |
|    | <b>R</b>             | Alto                 | Dinámico   | Interpretado | Estadística   |
|  | <b>Java</b>          | Alto                 | Estático   | Compilado*   | Big data, computación distribuida, herramientas gráficas                        |
|  | <b>Scala</b>         | Alto                 | Estático   | Compilado*   | Big data, computación distribuida   |
|  | <b>MATLAB/Octave</b> | Alto                 | Dinámico   | Interpretado | Computación numérica, simulación  |

# Rendimiento de una función

```
t0 = time()
ordenacionBurbuja(lista)
t1 = time()

print ("Lista ordenada:")
print(lista)

print ("Tiempo: {0:f} segundos".format(t1 - t0))
print ("Comparaciones:", numComparaciones)
```

```
Lista ordenada:
[0, 8, 16, 17, 35, 73]
Tiempo: 0.000011 segundos
Comparaciones: 15
None
33 function calls in 0.000 seconds
```



# Rendimiento de una función

```
import profile
from time import time

def ordenacionBurbuja(lista):
    'Variable global que puede se usada fuera de la función'
    global numComparaciones
    n = len(lista)

    for i in range(1, n):
        for j in range(n-i):
            numComparaciones += 1

            if lista[j] > lista[j+1]:
                'intercambio de valores'
                lista[j], lista[j+1] = lista[j+1], lista[j]

lista = [35, 16, 17, 73, 8, 0]
numComparaciones = 0

t0 = time()
ordenacionBurbuja(lista)
t1 = time()

print ("Lista ordenada:")
print(lista)

print ("Tiempo: {0:f} segundos".format(t1 - t0))
print ("Comparaciones:", numComparaciones)

profile.run('print(ordenacionBurbuja(lista))')
```

| ncalls      | totttime | percall | cumtime | percall | filename:lineno(function)                    |
|-------------|----------|---------|---------|---------|--|
| 18454895/33 | 4.665    | 0.000   | 4.665   | 0.141   | <stdin>:1(fib)                               |
| 33/1        | 0.000    | 0.000   | 4.665   | 4.665   | <stdin>:1(fib_seq)                           |
| 1           | 0.000    | 0.000   | 4.666   | 4.666   | <string>:1(<module>)                         |
| 1           | 0.000    | 0.000   | 4.666   | 4.666   | {built-in method builtins.exec}              |
| 1           | 0.000    | 0.000   | 0.000   | 0.000   | {built-in method builtins.print}             |
| 33          | 0.000    | 0.000   | 0.000   | 0.000   | {method 'append' of 'list' objects}          |
| 1           | 0.000    | 0.000   | 0.000   | 0.000   | {method 'disable' of '_lsprof.Profiler' obj} |
| 32          | 0.000    | 0.000   | 0.000   | 0.000   | {method 'extend' of 'list' objects}          |



[www.unir.net](http://www.unir.net)