# **Apuntes – Módulo 2**

# **Curso: Modeling and Simulation of Natural Processes**

**Universidad de Ginebra - Prof. Bastien Chopard et al.  
Semana 2 – Introducción a la programación con Python 3 y computación de alto rendimiento**

### **1. ¿Por qué programación en este curso?**

**Objetivo**: traducir fenómenos naturales a modelos matemáticos → métodos numéricos → implementación eficiente en ordenador.

* Los modelos complejos no siempre tienen solución analítica → necesitamos simulación numérica.
* Computación de alto rendimiento (HPC) = rapidez + precisión → esencial para modelos útiles.

### **2. Optimización del código**

Nos preguntamos acá ¿Importa cómo escribimos el código?

Se da un ejemplo que nos ayuda a responder:

Sumar un arreglo de 10⁶ enteros

* Lectura lineal: ~0.06s
* Lectura aleatoria: ~0.27s (muy ineficiente)
* sum() de Python: ~0.0065s
* NumPy: ~0.0006s
* C++: ~0.0005s

*Conclusión*:

* La forma del algoritmo y el lenguaje afectan fuertemente el rendimiento.
* NumPy se acerca mucho a C++ en eficiencia. Ideal para alto rendimiento en Python.

### **3. Conceptos de paralelismo**

Idea principal del video: dividir un problema entre varios procesadores para resolverlo más rápido (trabajaran en paralelo).

* Memoria compartida: varias CPUs acceden a una única memoria (usando OpenMP).
* Memoria distribuida: cada CPU tiene su propia memoria y se comunica por red (usando MPI).

No todo problema se puede paralelizar. Ejemplo: la serie de Fibonacci es secuencial.

### **4. Palabos – Solver paralelo de Lattice Boltzmann**

* Palabos es una biblioteca en C++ para simular flujos de fluidos con LBM.
* Totalmente paralela, sin necesidad de conocimientos en HPC.
* Usa una división del dominio espacial: cada bloque es simulado por una CPU diferente → se comunican a través de "interfaces".
* Ejemplo: flujo de aire en una oficina con aire acondicionado → simulado con cientos de núcleos.

### **5. Introducción a Python 3**

**¿Por qué Python 3?**

* Multiplataforma, sintaxis simple, eficiente con NumPy.
* Muchas bibliotecas científicas disponibles.
* Lenguaje interpretado → más fácil para desarrollo, portable.

Recomendación del curso: leer el libro *Dive into Python 3*.

### **6. Ejecutar un programa Python**

**Opciones de ejecución**:

1. **Shell interactiva** (python3)  
   * Ideal para probar código rápidamente.
   * Se puede usar como calculadora.
   * quit() o Ctrl+D para salir.
2. **Script .py**
   * Crear archivo (ej. mi\_script.py) con código.
   * Ejecutar en terminal con python3 mi\_script.py.
3. **Entorno Spyder** (IDE recomendado por el curso)  
   * Colores, autocompletado, ejecución directa con botón.

### **7. Variables y tipos de datos**

* Python no requiere declarar tipo → tipado dinámico (No debo decirle si pongo x = 5 que es un número, lo identifica el mismo).
* Una misma variable puede cambiar de tipo.
* Los identificadores distinguen entre mayúsculas y minúsculas (a ≠ A).

**Tipos principales**:

* int, float, string
* str (cadenas, inmutables)
* bool (True, False)
* None
* **Secuencias**:  
  + list: mutable
  + tuple: inmutable
  + range: secuencia de enteros

Con *type() puedo saber qué tipo tiene una variable*.

### **8. Operadores en Python**

* Aritméticos: +, -, \*, / (real), // (entera), %, \*\*
* Booleanos: <, <=, >, >=, ==, !=, and, or, not
* Sobre secuencias:  
  + + (concatenar), \* (repetir), [] (acceso), [:] (slice)
* Especiales: "a" in lista → verifica si elemento pertenece a secuencia

### **9. Sentencias condicionales (if, elif, else)**

Permiten ejecutar bloques de código según condiciones booleanas.

**Sintaxis básica:**

if condicion:

sentencias\_1

elif otra\_condicion:

sentencias\_2

else:

sentencias\_defecto

Se deja claro que ¡La indentación es obligatoria! Es lo que define qué pertenece a cada bloque.

### **10. Bucles (for, while)**

Estructuras que repiten acciones automáticamente.

#### **for loop**

Itera sobre cualquier secuencia (listas, rangos, strings...).

Ejemplo python:

for x in range(5):

print(x)

Se ejecuta una vez por cada elemento de la secuencia. Muy útil con listas, strings, arrays, etc.

### **11. Funciones (def)**

Permiten reutilizar código agrupando instrucciones bajo un nombre.

Sintaxis:

def nombre\_funcion(parametros):

"""Comentario opcional"""

sentencias

return resultado

* Una función puede retornar None si no se especifica return.
* Se pueden usar múltiples return dentro de una misma función.

Importante: si una función modifica un objeto mutable (como una lista), el cambio se mantiene fuera de la función también.

### **12. Uso de NumPy para optimización y álgebra lineal**

NumPy es una librería de Python especializada en operaciones numéricas rápidas, similar a Matlab.

**Ventajas**:

* Arrays multidimensionales (ndarray)
* Operaciones vectorizadas (sin bucles)
* Álgebra lineal, números aleatorios, transformadas, etc.

#### **Ejemplo de mejora de rendimiento:**

Comparando tiempo de suma de arrays grandes:

* Listas → ~1s
* NumPy → ~0.03s ¡Aproximadamente 30 veces más rápido!

## **Al final en los ejercicios se implementó un ejercicio práctico en Python – Integración numérica**

**Objetivo:** implementar una clase que integre la función f(x)=x2⋅sin⁡(x)⋅e−x en un intervalo [xmin,xmax]. (Adjunto en las carpetas)