ANÁLISIS NUMÉRICO I / ANÁLISIS NUMÉRICO – 2025
Guía básica de `Python`

Recomendaciones

- No usar espacios en los nombres de los archivos o carpetas

¡Comenzar a trabajar con `Python`!

- Abrir una terminal: `Crtl+Alt+T` o Aplicaciones....
- Ir al directorio donde están guardados los archivos: ∼`$ cd Documentos/Numerico_I`
- Abrir un editor de texto: ∼`$ gedit &`
- Abrir la terminal de `Python`: ∼`$ ipython` o ∼`$ python`
- Escribir una lista de números v:
  `In[n]:#>v = [1, 2.56, -3.43, 0]`.
- Conseguir un número aleatorio entre 0 y 1:
  `In[n]:#>import random`
  `In[n+1]:#>r = random.random()`.

Operaciones de Modificación

Las operaciones de modificación son:

`+`	sumas dos variables	`-`	restar dos variables
`*`	multiplicar dos variables	`/`	dividir dos variables
`**`	elevar una variable a la otra	`=`	asignarles un valor a una variable

Operaciones de Módulo y Cociente Entero

Las operaciones son:

`%`	operador módulo	`//`	operador cociente entero

Operaciones Lógicas

Las operaciones lógicas son:

`<=`	menor o igual a	`<`	menor que
`>`	mayor que	`>=`	mayor o igual a
`==`	igual a	`!=`	distinto

Funciones y librerias

- Funciones:
  Las funciones son un "pedazo de código" reutilizable que se crean con el objetivo de no repetir las mismas instrucciones dentro de un programa. Es decir, son bloques que nos permiten acortar el código no repitiendo estructuras que necesitamos utilizar varias veces.

  Una función es un conjunto de instrucciones empaquetadas bajo el mismo nombre. Es como un pequeño programa dentro del programa, una manera de crear nuestras propias instrucciones. Al igual que los programas, las funciones tienen un input y un output: el input son las variables de entrada y el output el valor o los valores que devuelven.

  En `Python`, las funciones se implementan de la siguiente forma:

```
def mi_funcion(a , b , c , ...):

    # Hago lo que necesite con las variables a, b, c, ...
    # En este caso, creamos las variables x, y, z, ...

    return x , y , z , ...
```

- Librerias:
  Una librería es código organizado en un conjunto de funciones implementadas por otras personas que nos facilitan realizar tareas, principalmente porque no debemos volver a programar este código.

  Para utilizar cualquier librería primero debemos importarla. Para ello, se usa la palabra clave import, pero hay muchas formas de emplearla. Sabiendo esto, una posibilidad es utilizar la siguiente sintaxis:

```
import (nombre de la libreria) as (nombre abreviado)
```

  De esta forma, estamos habilitando el uso de todo lo que contiene el paquete importado. El comando as nos permite indicar el nombre que le damos a la librería dentro de nuestro código.

Condicionales y bucles

- La estructura `if ...`
  La estructura del if simple es la siguiente:

```
if condición:
     Acciones a realizar si es cierta la condición
else:
     Acciones a realizar si es falsa la condición
```

  Observación: La indentación (tabs) marca el final de la estructura.

  Hagamos una función que verifica si un número n es divisible m y devuelve $True$ si lo es, $False$ en caso contrario.

```
def multiplo(n,m):

     if n % m == 0:
```

```
        print('Es divisible')
        return True
    else:
        print('No es divisible')
        return False
```

Observación: LLegar a un *return* desemboca en el final de la función.

- El bucle (loop) `for ...`

  En el bucle `for ...` , la ejecución de uno o varios comandos se repite un número fijo y predeterminado de veces.

```
for variable_contador in range(INICIO,FIN):
    Acciones_bucle_externo
    for variable_otro_contador in range(OTRO_INICIO,OTRO_FIN):
         Acciones_bucle_interno
```

  Por ejemplo, si se quiere imprimir los valores de una matriz $A \in \mathbb{R}^{m\times n}$, triangular superior, tal que $A_{i,j} = 1/(i+j)$ para $i \leq j$, se debe hace

```
def print_vander(m,n):
    for idx in range(1,m+1):
         for jdx in range(idx,n+1):
              print(f"A({idx},{jdx}) = {1/(idx+jdx)}")
```

- El bucle `while ...`

  Este bucle se usa cuando no se conoce el número de veces que debe repetirse la ejecución de cierto comando. Supongamos que queremos sumar números aleatorios hasta superar una tolerancia `Tol` y queremos saber cuantas veces iteramos; se debe hacer:

```
import random

def suma_aleatorio(Tol)

    s = 0
    contador = 0

    while s <= Tol:
        s = s + random.random()
        contador = contador + 1

   return s, contador
```