Clase Nº 2

Funciones

Ventajas de las funciones

- Permiten dividir el programa en módulos lógicos.
- Evitan código repetido.
- Pueden ser transportadas a otros proyectos.

Estructura de una función

- Toda función comienza con *def*.
- El encabezado de la función debe terminar con “dos puntos”.
- La sangría es obligatoria y define el alcance de la función.

Estructura de una función

```
def nombre(<parámetros>):
    """<Cadena de documentación>"""
    . . . . . .
    . . . . . .
    return <valor>
```

Estructura de una función

Reglas para los nombres de funciones:

- **Sólo se permiten letras, números y el guión bajo.**
- **No pueden comenzar con un número.**
- **No pueden coincidir con las palabras reservadas del lenguaje.**

Estructura de una función

- **El nombre de las funciones <u>debe tener sentido</u>.**
- **Deben evitarse nombres como "funcion", "nico" o "belu".**
- **Es recomendable que el nombre de la función incluya un verbo en infinitivo que describa su tarea.**

Estructura de una función

- **Los nombres de las funciones deben ser distintos de los nombres de las variables utilizadas en el programa y en la misma función.**

Estructura de una función

- **Las funciones deben escribirse *al principio* del programa.**
- **El programa principal debe comenzar *después* de la última función. ▼**
- **Se recomienda que el programa principal comience con un comentario que lo identifique.**

Estructura de una función

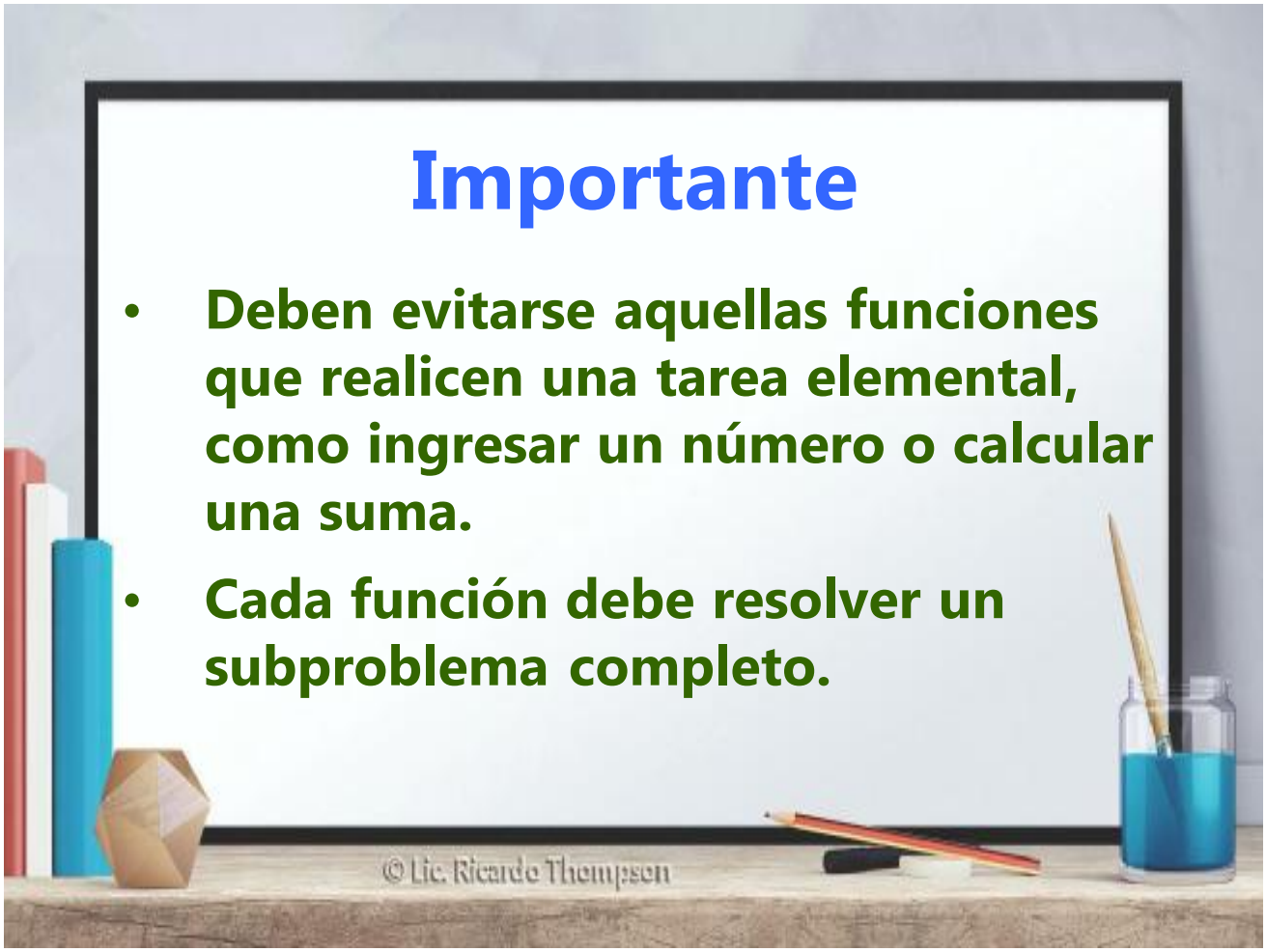
- La instrucción ***return*** termina la ejecución de la función y devuelve un valor a quien la haya llamado.
- Funciones que no retornen valores no necesitan llevar return.

Importante

- Cada función debe realizar ***una sola actividad***.
- No deben leerse ni imprimirse valores dentro de una función que realice otra tarea. ▼
- ***Jamás*** debe salirse de una función desde el interior de un ciclo. ▼

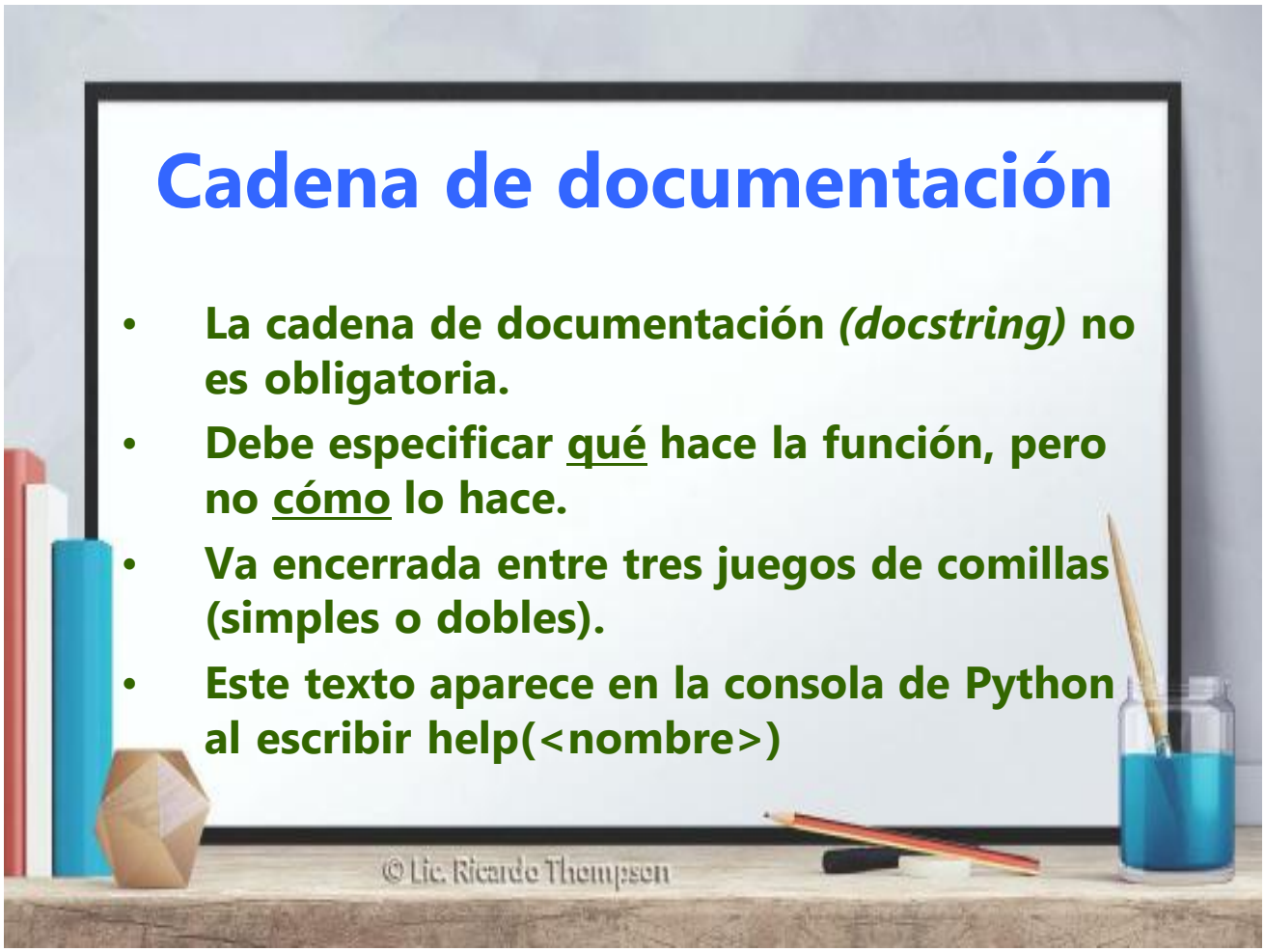
Importante

- Deben evitarse aquellas funciones que realicen una tarea elemental, como ingresar un número o calcular una suma.
- Cada función debe resolver un subproblema completo.



Cadena de documentación

- La cadena de documentación *(docstring)* no es obligatoria.
- Debe especificar qué hace la función, pero no cómo lo hace.
- Va encerrada entre tres juegos de comillas (simples o dobles).
- Este texto aparece en la consola de Python al escribir help(<nombre>)



Ejemplo 1

Desarrollar una función para calcular el factorial de un número entero positivo.


```
def calcularfactorial(n):
    """ Devuelve el factorial de un
        número entero positivo """
    fact = 1
    for i in range(1, n+1):
        fact = fact * i
    return fact

# Programa principal
a = int(input("Ingrese un número entero: "))
b = calcularfactorial(a)
print("El factorial es", b)
```

Variables locales

- Toda variable creada dentro de una función se considera *local*.
- Las variables del programa principal no pueden ser utilizadas dentro de una función, a menos que se pasen como parámetro . ▼

Variables locales

```
def calcularpromedio( ):
    total = (x + y) / 2
    return total

# Programa principal
x = int(input("Ingrese un numero entero: "))
y = int(input("Ingrese otro numero entero: "))
resultado = calcularpromedio( )
print("El promedio es", resultado)
```

INCORRECTO

Parámetros

- Los parámetros permiten que la función reciba datos para hacer su trabajo.
- Pueden pasarse 0 o más parámetros.
- Los paréntesis son obligatorios aunque no haya ningún parámetro, tanto en el encabezado como en la invocación de la función.

Parámetros

- Los parámetros ***formales*** son los que aparecen en el encabezado de la función, que es la línea que comienza con *def*.
- Los parámetros ***reales*** son los que se escriben en la llamada o invocación.
- Los parámetros formales actúan *en representación* de los parámetros reales.

Parámetros

Parámetros formales a y b

```
def calcularpromedio(a, b):
    total = (a + b) / 2
    return total

# Programa principal
x = int(input("Ingrese un numero entero: "))
y = int(input("Ingrese otro numero entero: "))
resultado = calcularpromedio(x, y)
print("El promedio es", resultado)
```

Parámetros reales x e y

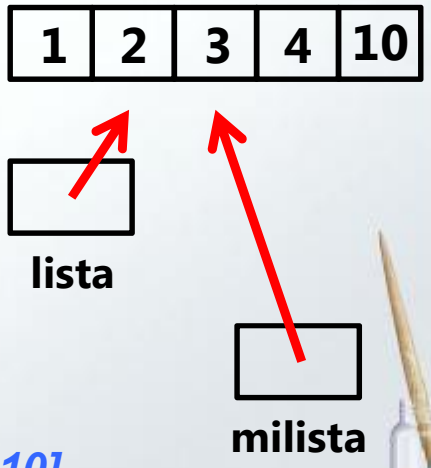
Parámetros

- Los parámetros se clasifican en ***mutables*** e ***inmutables***.
- Las variables simples, las cadenas de caracteres y las tuplas son ***inmutables***.
- Las listas, los conjuntos y los diccionarios son ***mutables***.

Parámetros mutables

```
def agregartotal(lista):
    total = 0
    for i in range(len(lista)):
        total = total + lista[i]
    lista.append(total)

# Programa principal
milista = [1, 2, 3, 4]
agregartotal(milista)
print(milista)      # [1, 2, 3, 4, 10]
```



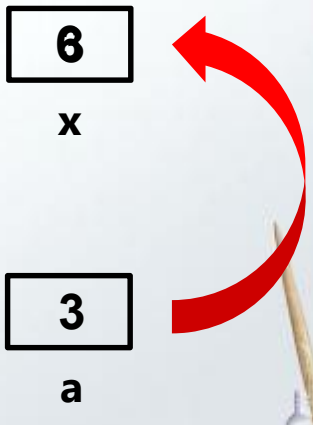
Parámetros mutables

- Si un parámetro formal <u>mutable</u> es modificado dentro de la función, el cambio afecta al parámetro real correspondiente.

Parámetros inmutables

```
def duplicar(x):
    x = x * 2

# Programa principal
a = 3
duplicar(a)
print(a)          # Imprime 3
```



Parámetros inmutables

- **Si la modificación se hace sobre un parámetro formal inmutable, el parámetro real no resulta afectado.**

Valor de retorno

- Python permite que las funciones devuelvan varios valores.
- Esto evita tener que devolver resultados a través de los parámetros, como ocurre en otros lenguajes de programación.

Ejemplo 2

Cómo devolver más de un valor de retorno

Escribir una función para ingresar una fecha por teclado.

```
def leerfecha( ):
    """ Lee una fecha por teclado
        y devuelve tres enteros """
    dia = int(input("Dia ? "))
    mes = int(input("Mes ? "))
    año = int(input("Año ? "))
    return dia, mes, año

# Programa principal
d, m, a = leerfecha( )
print(d, "/", m, "/", a, sep="")
```


Parámetros

- Los parámetros pueden tener ***valores por omisión.***
- Ante la ausencia de algún parámetro se utiliza el valor por omisión.
- Ésto permite invocar a una función con menos parámetros de los previstos.

Ejemplo 3

Uso de parámetros por omisión

Escribir una función para calcular la raíz n-ésima de un número.


```
def calcularraiz(radicando, indice=2):
    return radicando ** (1/indice)

# Programa principal
a = float(input("Ingrese el radicando: "))
r2 = calcularraiz(a)
r3 = calcularraiz(a, 3)
print("Raiz cuadrada:", r2)
print("Raiz cúbica", r3)
```

Parámetros

- Los parámetros también pueden pasarse por nombre, en lugar de hacerlo por posición.
- Los parámetros con nombre deben escribirse ***después*** de los pasados por posición,

Ejemplo 4

Uso de parámetros con nombre

```
def calcularraiz(radicando, indice=2):
    return radicando ** (1/indice)

# Programa principal
r5 = calcularraiz(indice=5, radicando=32)
print("Raíz quinta:", r5)
print(calcularraiz(27, indice=3))
```


Funciones Lambda

- Son funciones pequeñas, anónimas, desechables y de una sola línea.
- Se pueden usar en cualquier contexto donde se admita una función y viceversa.
- Las funciones lambda se escriben en el lugar donde se necesitan.

Funciones Lambda

```
cuadrado = lambda x: x**2
print(cuadrado(3))   # imprime 9
```

...que es equivalente a:

```
def cuadrado(x):
    return x**2

# Programa principal
print(cuadrado(3))
```


Funciones Lambda

Sintaxis:

<var> = lambda <params> : <valor de retorno>

La variable ubicada a la izquierda del signo igual pasa a comportarse como una función.

Funciones Lambda

Ejemplo con dos parámetros:

```
raiz = lambda x, y : x**(1/y)
a = raiz(25, 2)          # devuelve 5
print(raiz(8, 3))        # imprime 2
```


Funciones Lambda

También pueden usarse parámetros con valores por omisión:

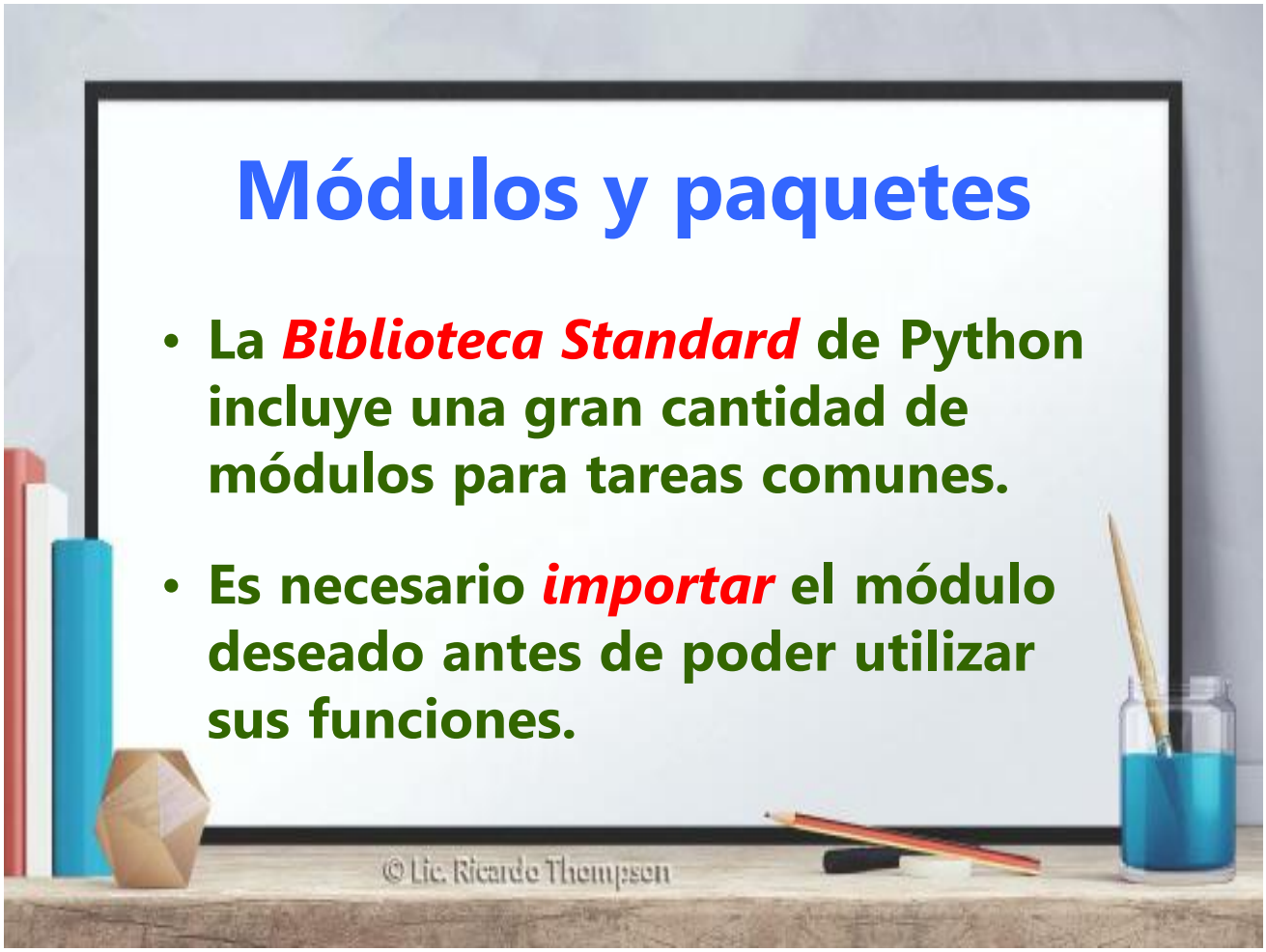
```
raiz = lambda x, y=2 : x**(1/y)
print(raiz(81))   # imprime 9
```

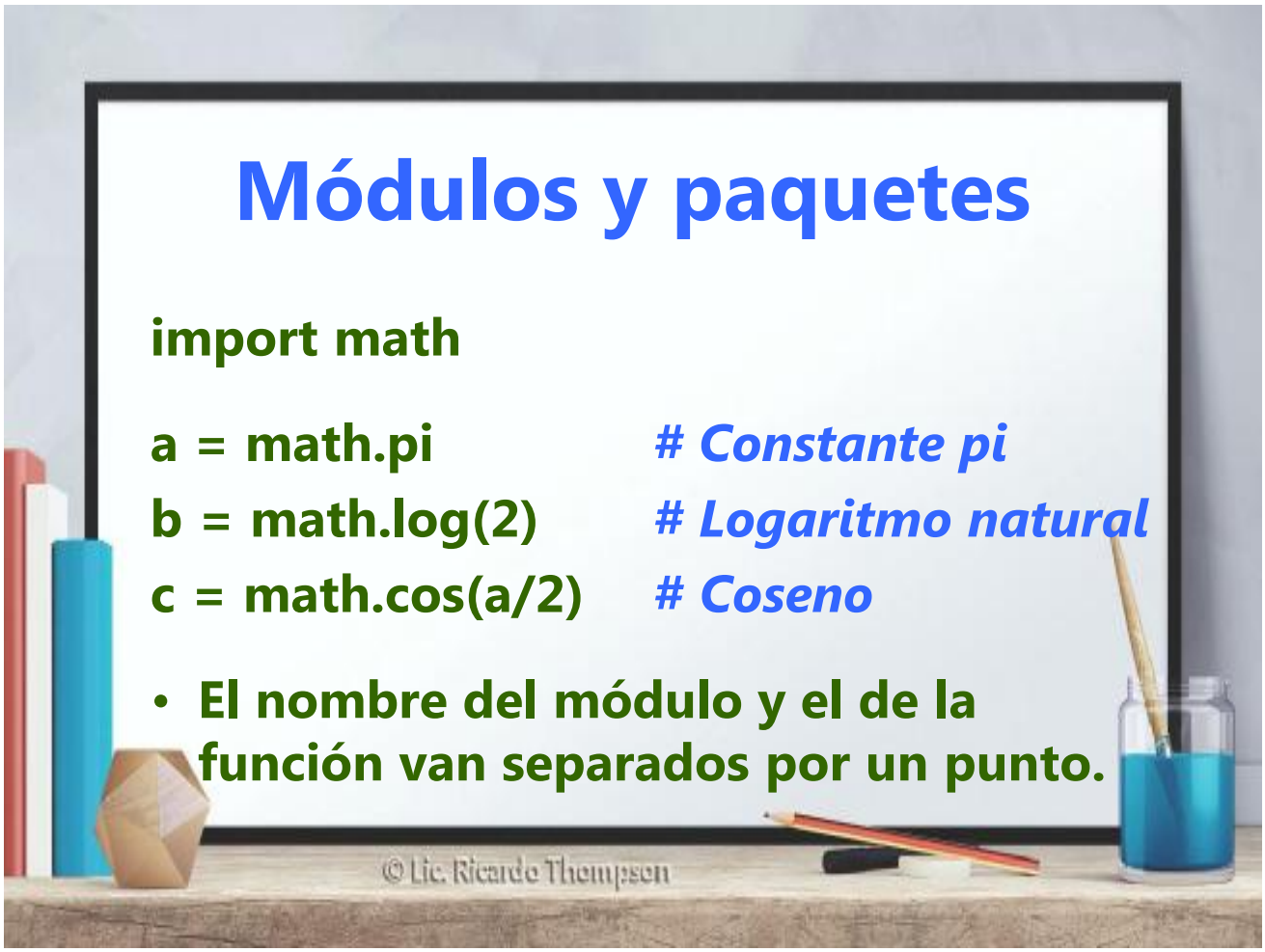
Funciones Lambda

- Las funciones lambda se agregaron a Python por compatibilidad con el lenguaje Lisp.
- Su utilidad será más evidente al aplicarlas a listas, combinándolas con map( ), filter( ), sort( ), etc.

Módulos y paquetes

- Un ***módulo*** es un conjunto de funciones.
- Un ***paquete*** es un conjunto de módulos.
- Equivalen a las bibliotecas de otros lenguajes.


Módulos y paquetes
• La *Biblioteca Standard* de Python incluye una gran cantidad de módulos para tareas comunes.
• Es necesario *importar* el módulo deseado antes de poder utilizar sus funciones.
© Lic. Ricardo Thompson


Módulos y paquetes
import math
a = math.pi # Constante pi
b = math.log(2) # Logaritmo natural
c = math.cos(a/2) # Coseno
• El nombre del módulo y el de la función van separados por un punto.
© Lic. Ricardo Thompson

Módulos y paquetes

Además de math, otros módulos de uso frecuente son:

- **random: Números al azar.**
- **os: Funciones del sistema operativo.**
- **tkinter: Interfaces visuales.**
- **sqlite3: Manejo de bases de datos.**

Funciones incorporadas

Algunas funciones forman parte del intérprete Python, por lo que no es necesario importar ningún módulo.

- **abs( ): Valor absoluto**
- **len( ): Longitud**
- **int( ): Convierte a número entero**
- **float( ): Convierte a número real**


Ejercitación
• Práctica 1: Completa
© Lic. Ricardo Thompson