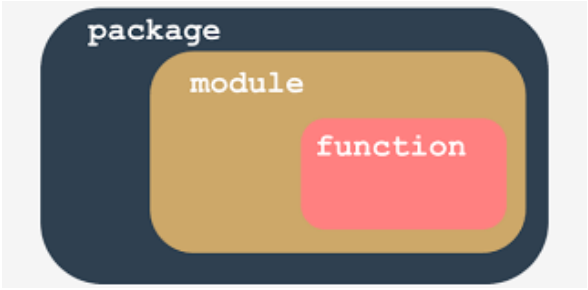
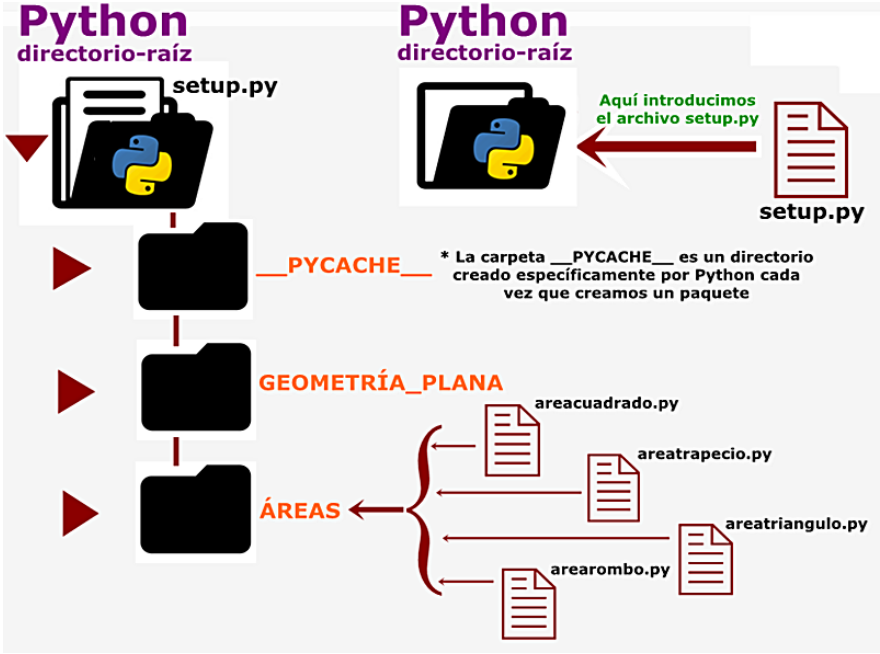
UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD DE INGENIERÍA

Cursos Básicos
Primer Ciclo 2025

Fundamentos de Programación

Funciones Módulos y Paquetes





¿Qué veremos hoy?

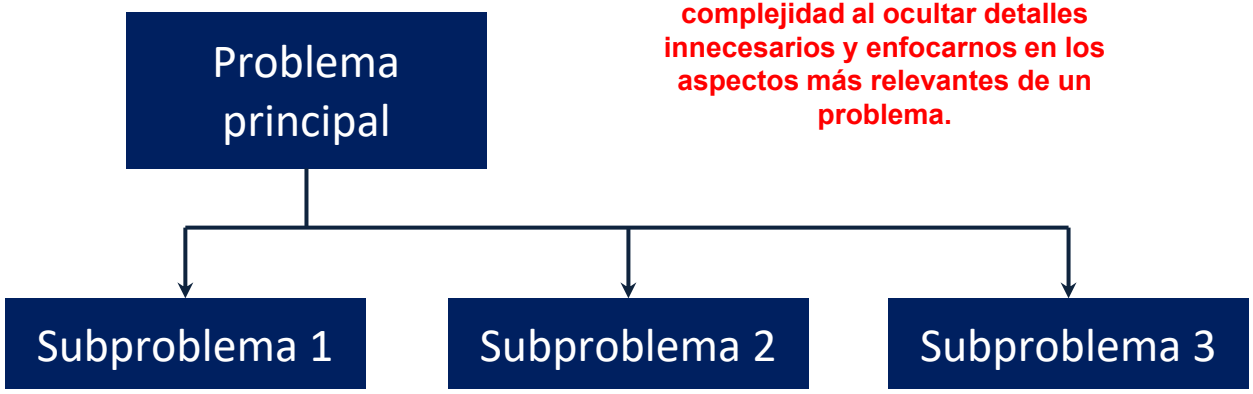
- Abstracción
- Procedimientos y Funciones
- Parámetros, Valores y Ámbito de Variables
- Manejo de módulos

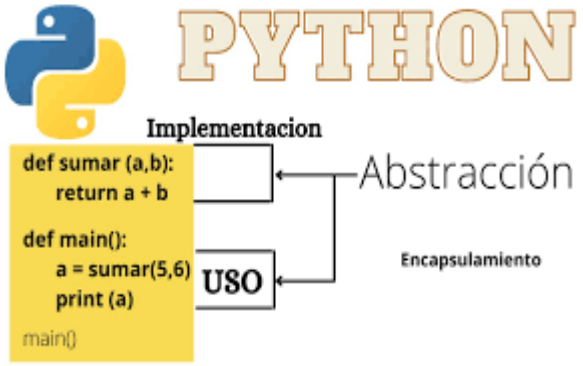
Abstracción

¿Qué es la abstracción?

- La abstracción es un concepto fundamental en programación que implica la simplificación y la creación de modelos que representan la realidad de manera más simple y manejable.

La abstracción nos permite manejar la complejidad al ocultar detalles innecesarios y enfocarnos en los aspectos más relevantes de un problema.





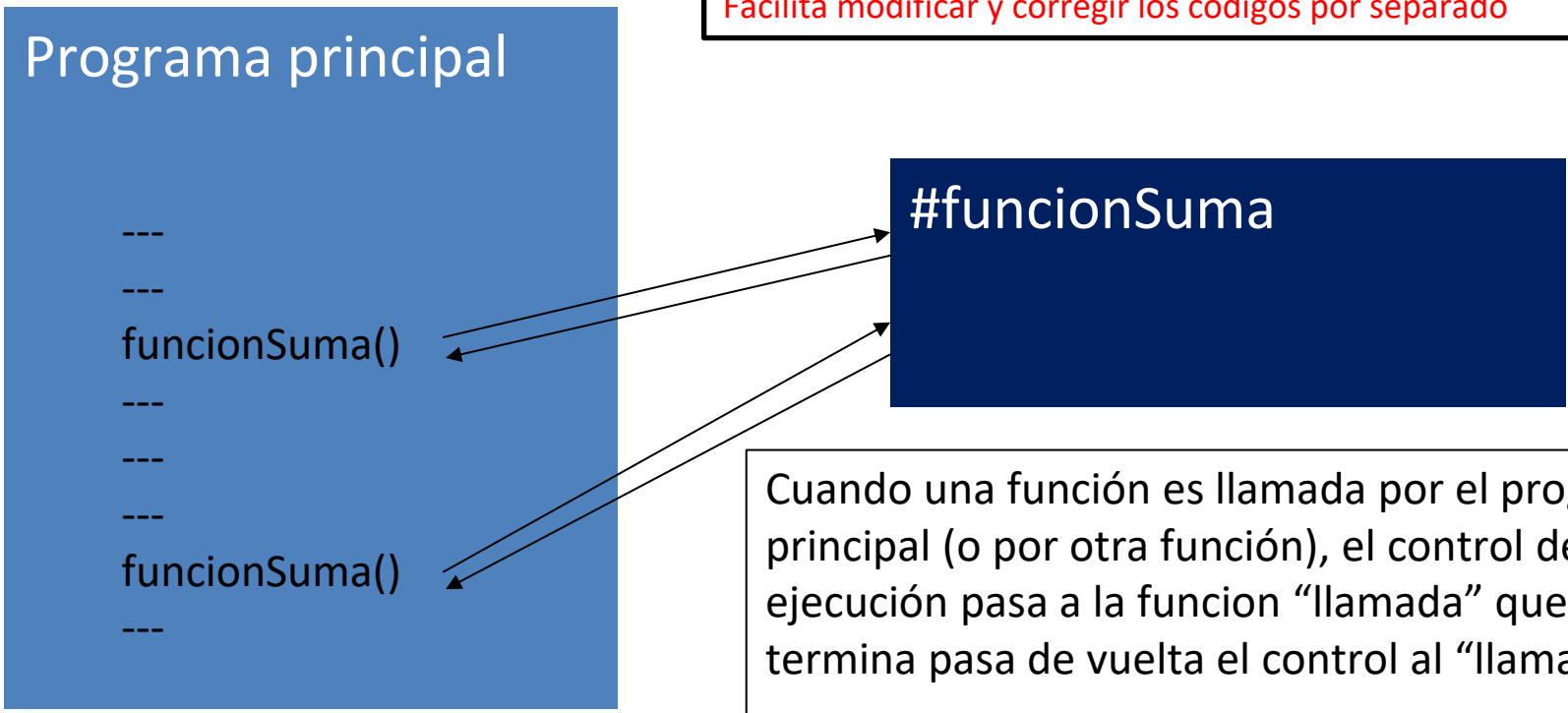
Los Subprogramas facilitan la Abstracción

- Los subprogramas nos ayudan a dividir tareas complejas en problemas más pequeños y manejables.
- Esto permite una mejor organización del código y una mayor reutilización de la lógica.

Funciones

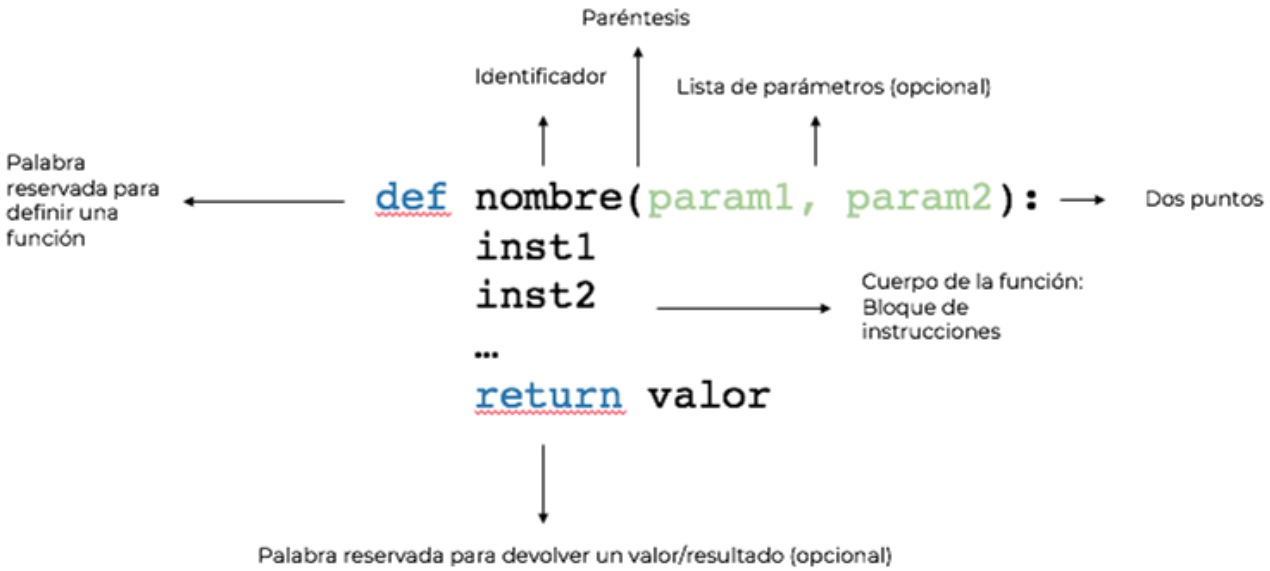
Las funciones están diseñadas para ejecutar alguna tarea específica, se escriben ***solamente una vez***, pero pueden ser referenciadas en ***diferentes puntos*** del programa, de modo que se puede evitar la duplicación innecesaria de código.

Programas más claros, legibles y menos complejos
Reutilizar código, subprogramas, módulos
Facilita modificar y corregir los códigos por separado



Cuando una función es llamada por el programa principal (o por otra función), el control de la ejecución pasa a la funcion “llamada” que cuando termina pasa de vuelta el control al “llamador”.

Definición y llamado



Creando o definiendo

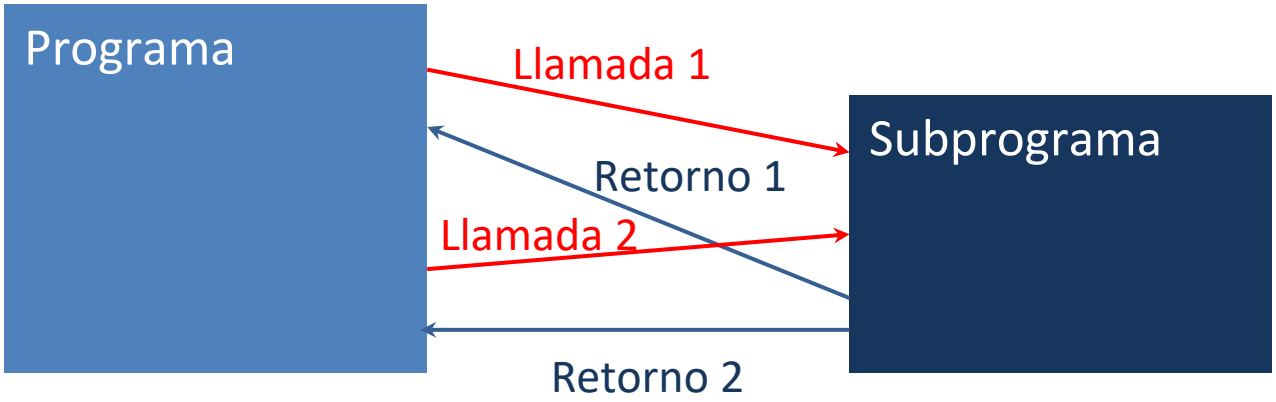
```
# Definición de una función
def suma(a, b):
      return a + b
```

Llamando (invocando)

```
# Llamadas a los subprogramas
resultado = suma(3, 5)
print("La suma es:", resultado)
resultado = suma(4, 3)
print("La suma es:", resultado)
```

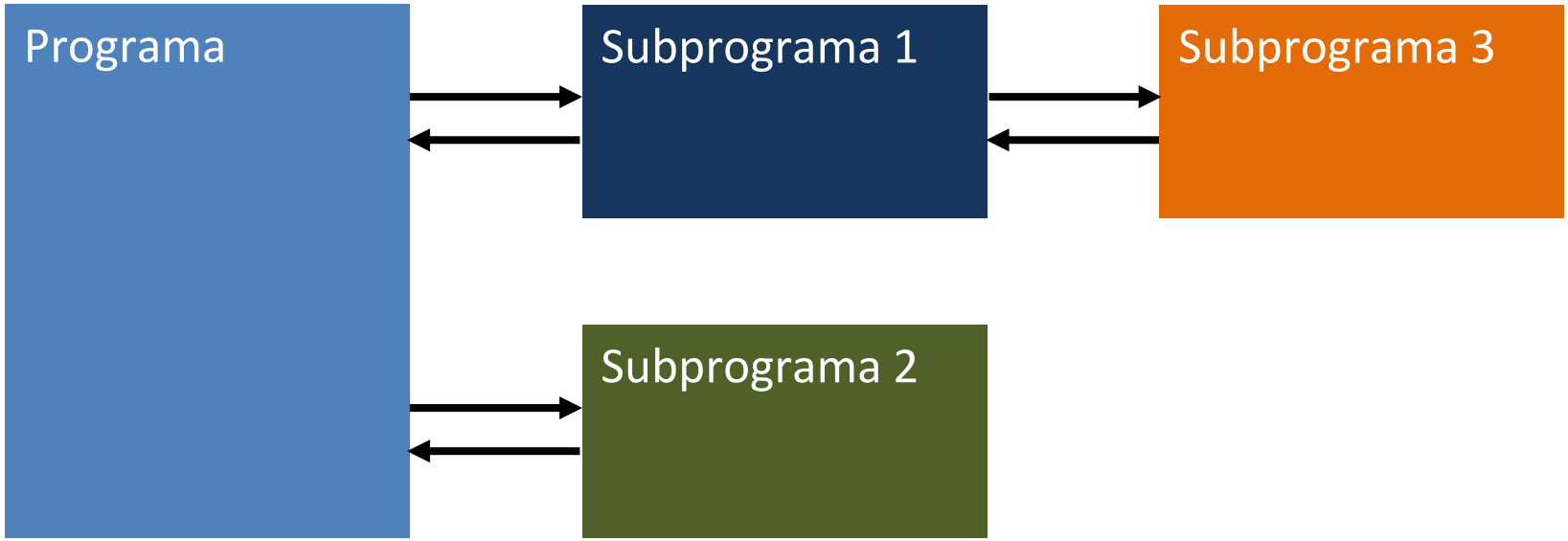
Invocación a subprogramas

El programa principal puede **llamar** o **invocar** a un subprograma en diferentes lugares. Cada vez que es llamado, este subprograma ejecuta una tarea, y a continuación **devuelve** el control al programa desde el lugar donde se hizo la invocación.



Invocación a subprogramas

Un subprograma puede llamar a su vez a sus propios subprogramas.



Procedimientos vs Funciones

- Los procedimientos (no devuelven resultados)
- Los valores que recibe una función (procedimiento) son los parámetros.

```
# Definición de un procedimiento

def saludar(nombre):
      print("¡Hola! "+nombre)

saludar("Diego")
saludar("Juan")
saludar("Pedro")
```

Los parámetros (argumentos) se especifican después del nombre de la función, dentro del paréntesis. Puedes agregar tantos argumentos como quieras, simplemente sepáralos con una coma.

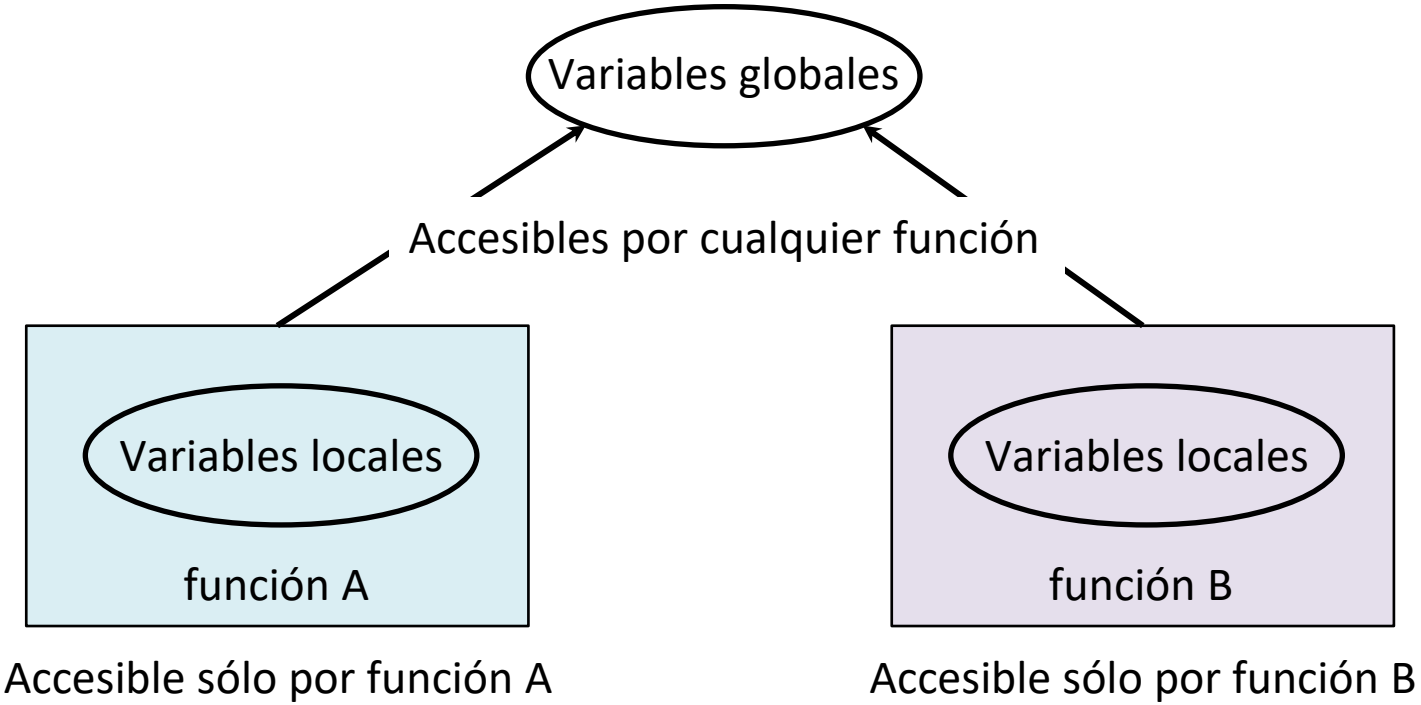
•Casos de Uso:

- Los procedimientos son útiles para acciones que no necesitan devolver un valor, como imprimir en pantalla.
- Las funciones son útiles cuando necesitamos calcular y devolver un resultado específico.

Ámbito de las variables

Variables locales: definidas en funciones y bloques. Sólo pueden emplearse (o son *visibles*) dentro de la función/bloque.

Variables globales: definidas fuera de las funciones, generalmente al principio. Accesible en todas las funciones.



Ámbito de las variables

Normalmente, cuando creas una variable dentro de una función, esa variable es local y solo se puede usar dentro de esa función.

global

```
texto= "Cool"
def imprimir(mensaje):
    print(mensaje+texto)
imprimir("Python es")
```

Que pasaría?

```
def sumar(x):
    x=x+1
x=10
sumar(x)
print(x)
```

Que pasaría?

```
x=0
def sumar():
    x=x+1
sumar()
print(x)
sumar(x)
print()
```

Hay que avisarle que modificaremos algo global

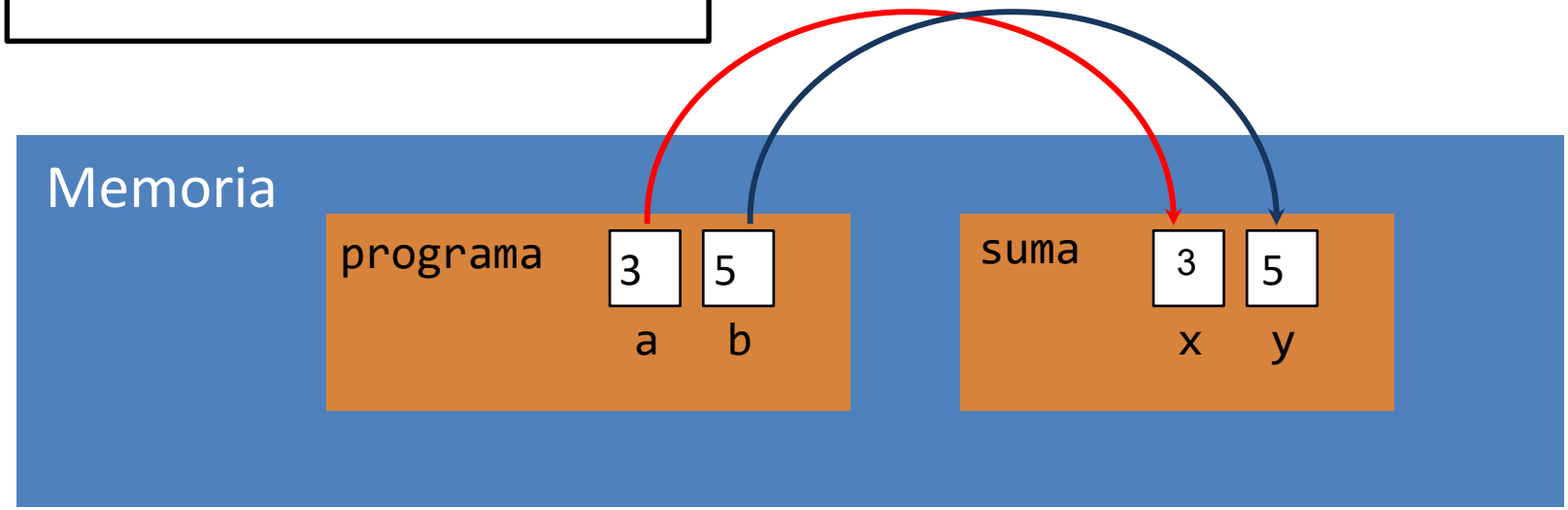
```
x=0
def sumar():
    global x
    x=x+1
sumar()
print(x)
sumar(x)
print()
```

```
UnboundLocalError: local variable 'x' referenced before assignment
```

Paso por Valor

En el paso por valor, se pasa una copia del valor de la variable al subprograma. Los cambios realizados dentro del subprograma no afectan a la variable original.

```
# Definición de una función
def suma(x, y):
    return x + y
# Llamadas a los subprogramas
a=3
b=5
resultado = suma(a, b)
print("La suma es:",
resultado)
```



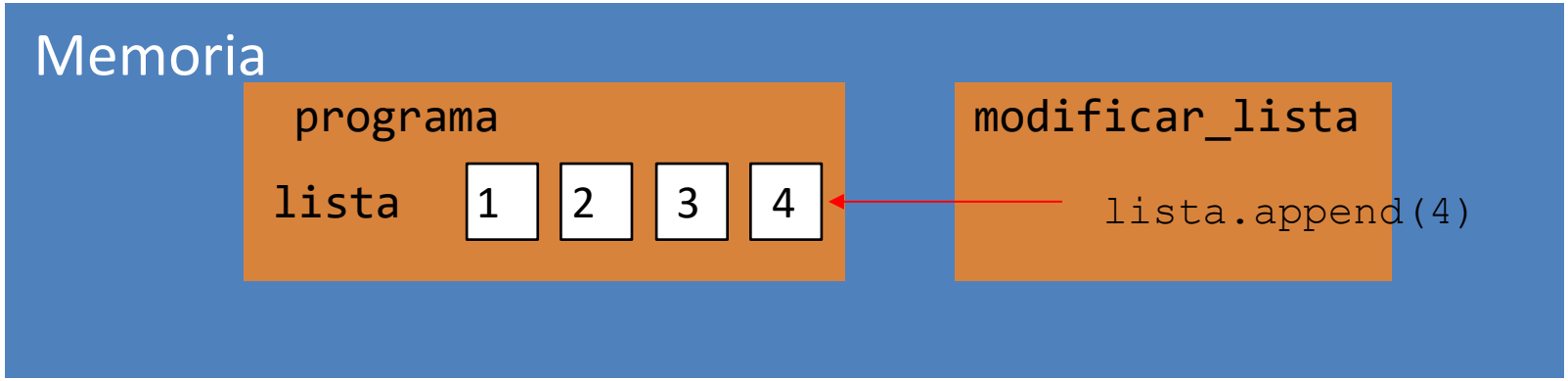
Paso por Referencia

En el paso por referencia, se pasa la referencia a la variable al subprograma. Los cambios realizados dentro del subprograma afectan a la variable original

En Python, las variables de tipo compuesto, como las listas, los diccionarios y los objetos, se pasan por referencia cuando se utilizan como argumentos de una función o método.

```
def modificar_lista(lista):
    lista.append(4)


mi_lista = [1, 2, 3]
modificar_lista(mi_lista)
print("Paso por Referencia:", mi_lista)
# La lista original se modifica
```



Objetivos del uso de subprogramas

- Modularización de programas: facilita la resolución de subproblemas en forma independiente al programa principal.
- Evita la repetición de código: permite parametrizar soluciones comunes. Se ahorra espacio de memoria para datos (variables) y para código.
- Permite ordenar códigos extensos de manera a facilitar sobre todo la lectura y el mantenimiento de los programas.

Un subprograma puede realizar entonces las mismas acciones que un programa: aceptar datos, realizar cálculos y devolver resultados. Un subprograma, sin embargo, se utiliza por el programa principal con un propósito **específico**.

Nota: pese a que son diferentes en definición, se suele usar indistintamente subprograma o función.

Funciones de funciones

$$C_{n,m} = \frac{n!}{m! * (n-m)!}$$

```
def factorial(n):
    fac=1;
    for i in range(1,n+1):
       fac*=i
     return fac
```

```
def combinacion( m, n):
    c = factorial(n)/(factorial(m)*factorial(n-m))
    return c
```

```
combinacion(m,n)
```

$n!$

Funciones lambda

Una función lambda en Python es una función anónima y pequeña que se define utilizando la palabra clave lambda. A diferencia de las funciones regulares definidas con def, las funciones lambda pueden tomar cualquier número de argumentos, pero solo pueden contener una expresión.

```
sumar = lambda x, y: x + y
print(sumar(3, 5))  # Salida: 8
```

```
cuadrado = lambda x: x ** 2
print(cuadrado(4))  # Salida: 16
```

```
es_par = lambda x: x % 2 == 0print(es_par(5))  # Salida:
Falseprint(es_par(6))  # Salida: True
```

Utilización de Funciones Externas

Utilizamos la palabra clave **import** seguida del nombre del módulo que contiene la función que deseamos utilizar.

```
import modulo
```

```
# Ejemplo de Importación y Utilización de Funciones Externas
import math
# Utilizar la función sqrt() del módulo math
raiz = math.sqrt(16)
print("La raíz cuadrada de 16 es:", raiz)
```

```
# Ejemplo de Funciones Externas Útiles
import random
# Generar un número aleatorio entre 1 y 100
numero_aleatorio = random.randint(1, 100)
print("Número Aleatorio:", numero_aleatorio)
```

Funciones útiles en `math`

Ejemplo: imprimir la raíz cuadrada de π, y los valores de las funciones seno, coseno y tangente de $\pi/3$.

```
import math #contiene las funciones matemáticas

pi = math.pi
print("Valor de pi:",pi)
print("Raiz cuadrada de pi:",math.sqrt(pi))
print("\nAlgunas funciones trigonometricas:")
print("sen(pi/3) =",math.sin(pi/3))
print("cos(pi/3) =",math.cos(pi/3))
print("tan(pi/3) =",math.tan(pi/3))
```

Funciones útiles en `math`

Funciones trigonométricas:

- `sin(x)`: función seno
- `cos(x)`: función coseno
- `tan(x)`: función tangente
- `asin(x)`: función arcoseno
- `acos(x)`: arcocoseno
- `atan(x)`: función arcotangente

Funciones exponenciales y logarítmicas:

- `exp(x)`: función exponencial e^x
- `log(x)`: logaritmo natural
- `log10(x)`: logaritmo en base 10
- `log2(x)`: logaritmo en base 2

Funciones hiperbólicas:

- `sinh(x)`: función seno hip.
- `cosh(x)`: función coseno hip.
- `tanh(x)`: función tangente hip.
- `asinh(x)`: función arcoseno hip.
- `acosh(x)`: arcocoseno hip.
- `atanh(x)`: función arcotangente hip.

Funciones de redondeo:

- `ceil(x)`: redondea hacia arriba.
- `floor(x)`: redondea hacia abajo.

https://docs.python.org/3/library/math.html

Números aleatorios – Enteros

La función `randint()` se encuentra en la librería/módulo `random`, y su sintaxis es la siguiente:

```
v1 = random.randint(inicio, fin)
```

Esta función devuelve un número entero entre `inicio` y `fin` (incluyendo estos límites).

```
import random
for i in range(10):
    x = random.randint(0,100)
    print(x,end=" ")
```

```
8 46 15 62 69 84 55 11 54 66
```

https://docs.python.org/3/library/random.html

Números aleatorios – Reales (*float*)

La función `uniform()` se encuentra en la librería/módulo `random`, y su sintaxis es la siguiente:

```
v1 = random.uniform(inicio, fin)
```

Esta función devuelve un número real (de punto flotante) entre `inicio` y `fin` (incluyendo generalmente estos límites).

```
import random
for i in range(10):
    x = random.uniform(0,100)
    print(f"{x:.3f}",end=" ")
```

```
1.453 23.797 1.667 42.304 57.314 44.467 61.721 0.171 11.329 37.357
```

https://docs.python.org/3/library/random.html

Mi modulo personalizado

Qué son los Módulos en Python y Por qué Son Útiles

Los módulos son archivos de Python que contienen definiciones y declaraciones, como variables, funciones y clases, que pueden ser utilizadas en otros programas Python.

Utilidad de los Módulos:

Los módulos permiten la organización y reutilización de código.

Facilitan la colaboración y el mantenimiento del código al dividirlo en partes lógicas y separadas.

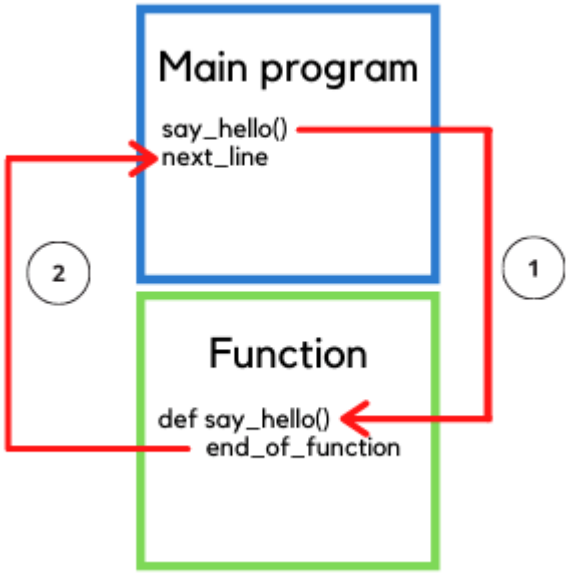
 # Ejemplo de Creación y Utilización de un Módulo Personalizado

```
# Contenido del archivo mi_modulo.py
def saludar(nombre):
    print("¡Hola,", nombre, "!")
```

```
#Utilización del módulo personalizado
import mi_modulo
mi_modulo.saludar("Juan")
```

Puntos Importantes a Revisar

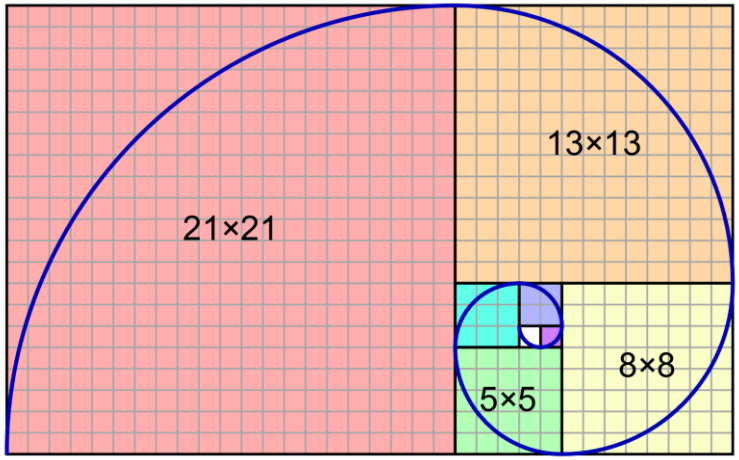
- Subprogramas: concepto y características
- Funciones y procedimientos
- Subprogramas en Python
- Uso de `return`
- Ámbito de las variables
- Paso de parámetros
- Funciones útiles en `math`
- Generación de números aleatorios

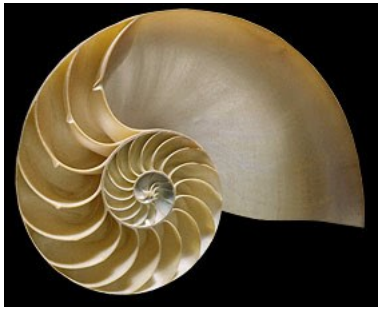


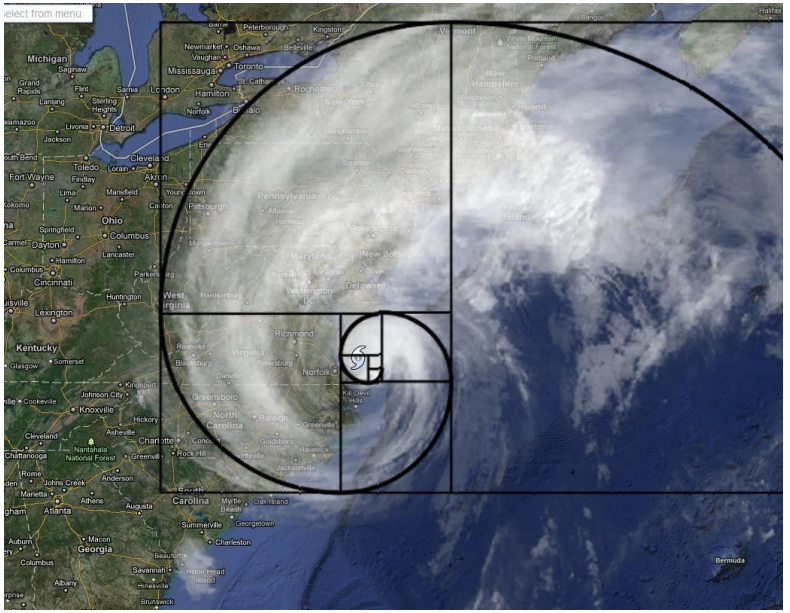
Ejemplos

Ejemplo 3: Calcular el enésimo término de la sucesión de Fibonacci, donde cada término es la suma de los dos anteriores, y sabiendo que los dos primeros términos son 0 y 1.

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…
```







Recursividad

Series de fibonacci

```
def fibonacci (n):
    ant1 = ant2 = 1;
    if ((n == 0) or (n == 1)):
       actual = 1;
    else:
        for i in range(2,n+1) :
            actual = ant1 + ant2
            ant2 = ant1
            ant1 = actual
         return (actual)
```

Ejercicio Propuesto

$$\sum a^n$$

$$1 + a + a^2 + a^3 + a^4 + \cdots$$