Programación Básica
Clase 08

Agenda

Sub programas
- Funciones
- Procedimientos
- Parámetros, valores de retorno
- Ámbito de variables

Introducción a los sub programas

La división de tareas es una excelente alternativa para mejorar los tiempos de producción, rendimiento, calidad, etc.

Analicemos el caso de la construcción de un edificio en donde tenemos 20 trabajadores para terminarlo y es necesario que tengamos división de tareas para:

- Obtener los permisos
- Creación de los planos
- Instalación los componentes eléctricos
- Uso de la maquinaria
- Construcción del edificio

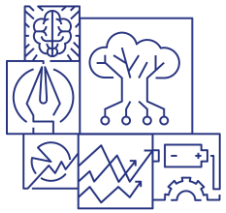
No todos pueden saber de todo, tendremos personal especializado para hacer tareas en momentos específicos. Probablemente el personal de soldadura no tiene por qué conocer sobre los presupuestos para el material.

fidélitas Universidad

Introducción a los sub programas

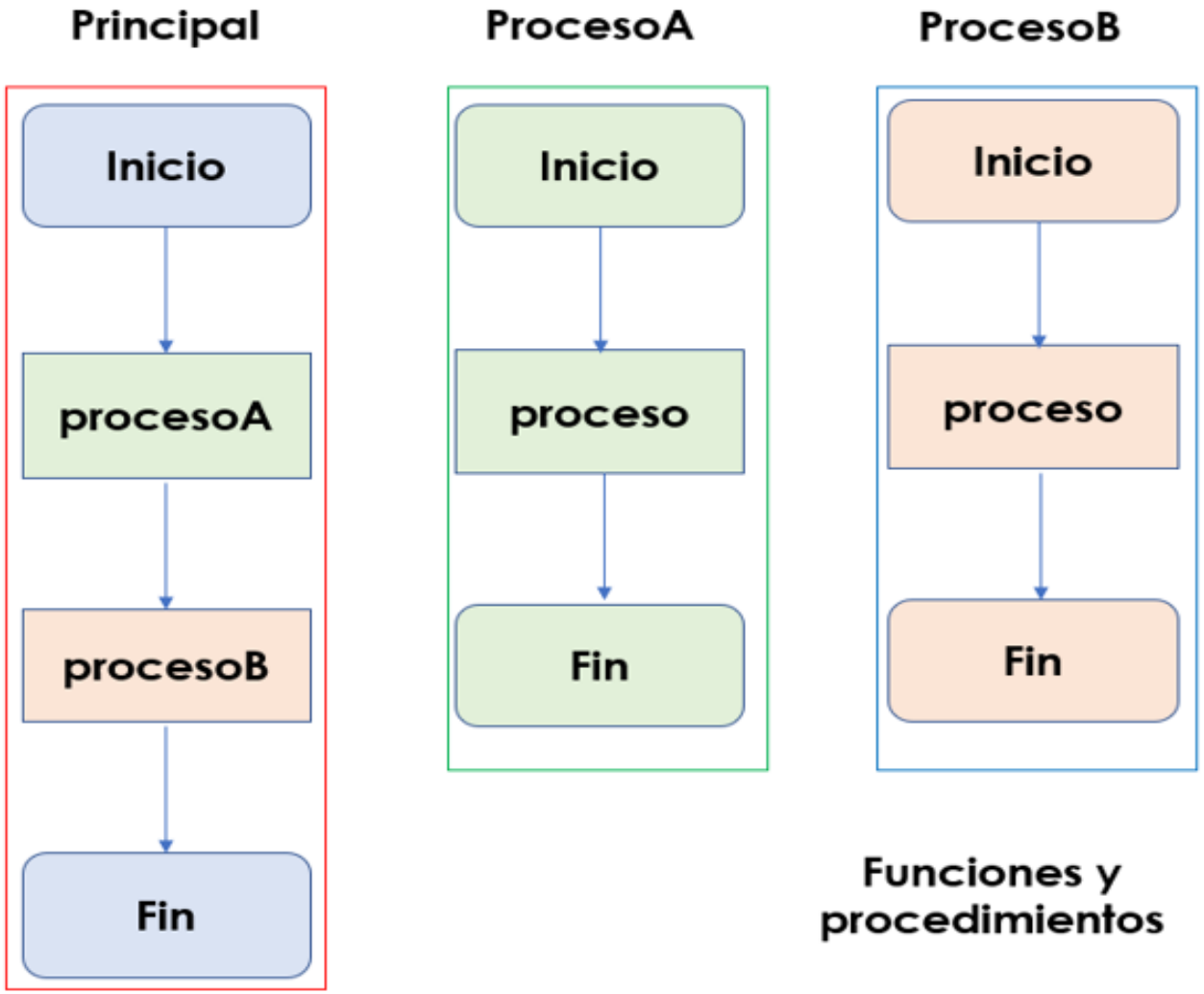
En la programación tenemos una situación similar, en la medida que los programas crecen, podremos identificar varias tareas y permitir que un solo bloque realice todas las tareas es **complejo**, **propenso a errores** y **difícil de dar mantenimiento**.

Aquí es cuando encontramos como solución dividir nuestro programa en partes que cumplan labores específicas.



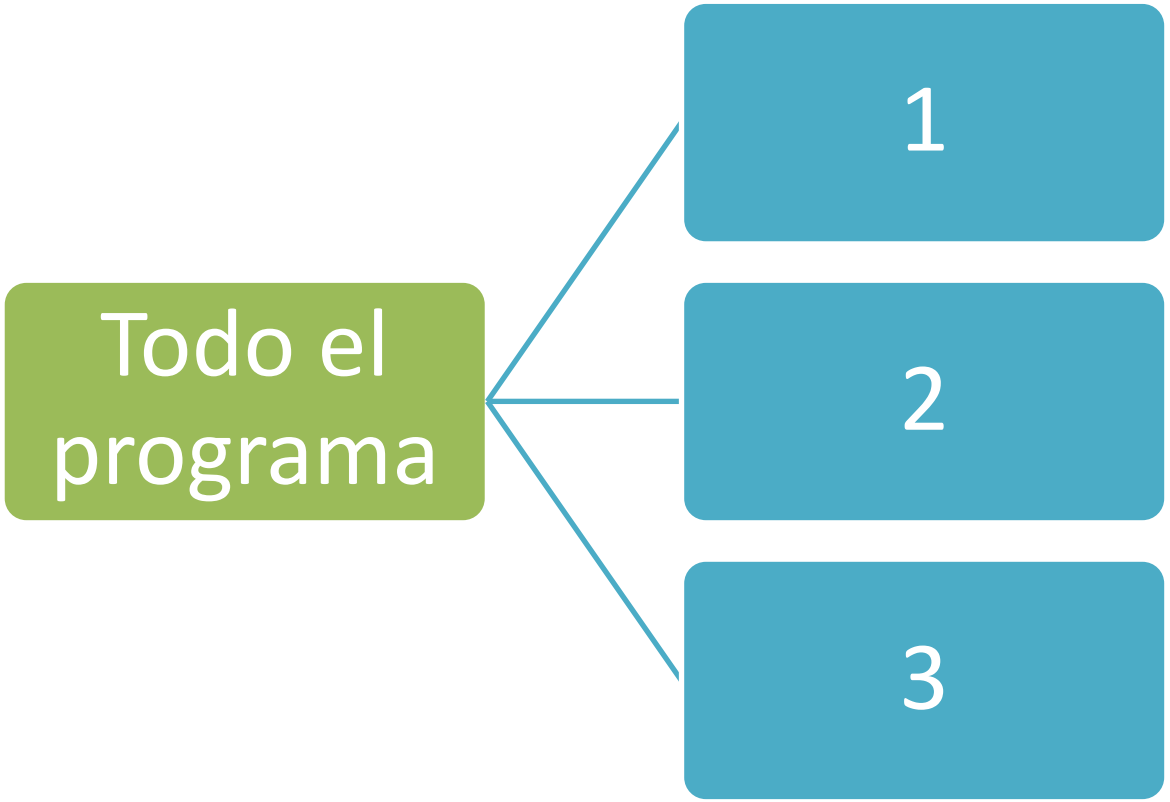
fidélitas Universidad

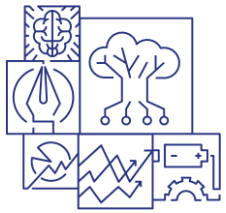
Introducción a los sub programas



Funciones y procedimientos

La división de tareas en varios sub programas sigue el criterio de lograr **reducir el funcionamiento de un bloque de código a una labor exclusiva e independiente** o al menos **tan pequeña que me permita enfocarme en su desarrollo y olvidarme del resto de manera temporal.**





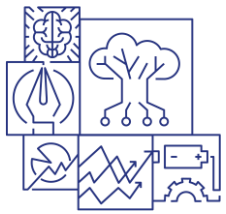
fidélitas Universidad

Python no tiene procedimientos, tiene funciones

A diferencia de otros lenguajes, Python no cuenta con procedimientos (subprograma que no devuelve ningún valor).

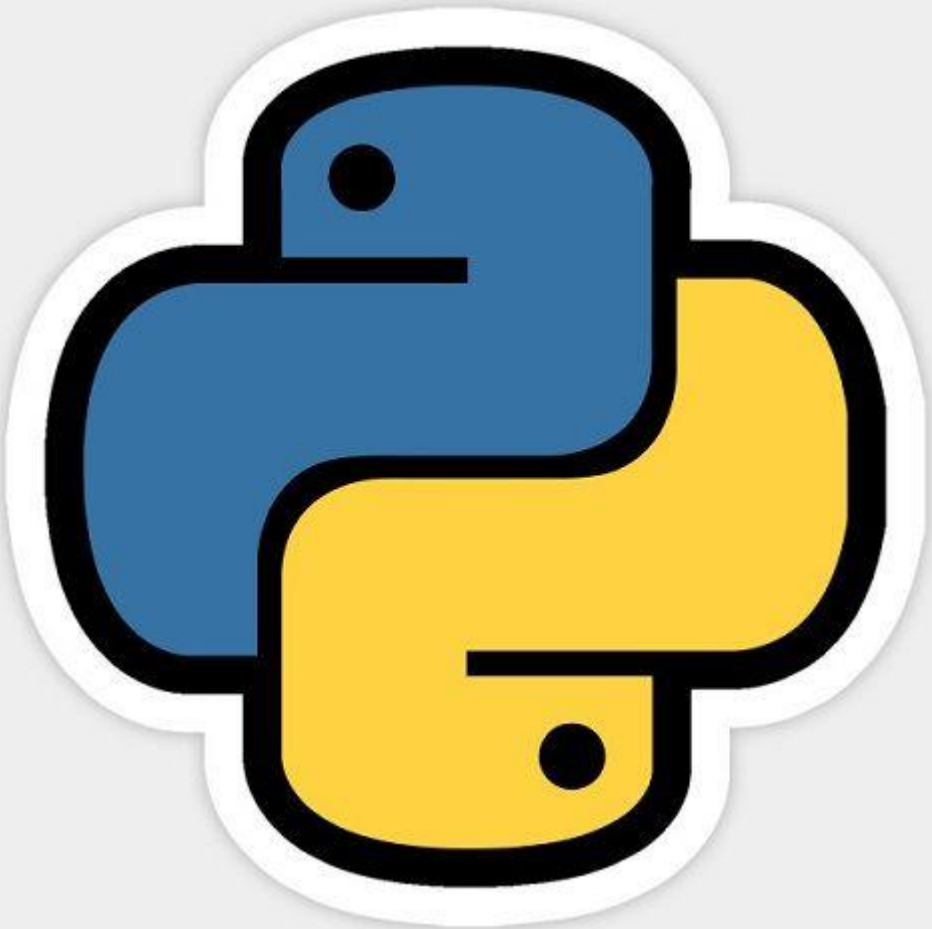
En Python, los subprogramas siempre devolverán un valor. Es por esta razón que reciben el nombre de funciones.

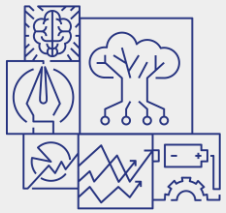
Se afirma lo anterior debido a que, aunque puede ocurrir que la palabra return no aparezca explícitamente en el cuerpo de la función, Python, internamente devolverá por defecto el valor None.



fidélitas Universidad

Tipos de funciones en Python



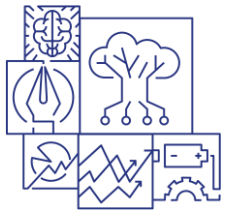


fidélitas
Universidad

Función sin retorno (Retorno *None*)

```
def saludar():
    nombre="María"
    print("¡Bienvenida a Python,",nombre,"!")
```

Se ejecutan las instrucciones del cuerpo de la función y no devuelve ningún valor al punto del llamado de la función.



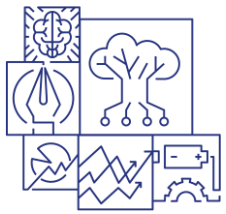
fidélitas Universidad

Función con retorno

```
def multiplicar():
    valor1=10
    valor2=20
    return valor1*valor2
```

Se ejecutan las instrucciones del cuerpo de la función y a través de la palabra reservada return se devuelve un valor al punto del llamado de la función.

Nota: Las funciones con retorno siempre retornarán un valor y no más de un valor.



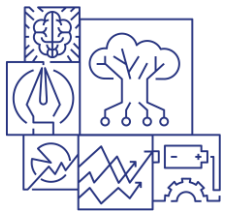
fidélitas Universidad

¿Cómo lo implementamos en Python?

```
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b
print("La suma es: ", suma)
print("La resta es: ", resta)
print("La multiplicación es: ", multiplicacion)
print("La división es: ", division)
```

Consideremos este programa, solicita 2 números y nos entrega el resultado de las 4 operaciones básicas

A pesar de que es bastante simple, consideremos que hace 4 tareas muy claras, una por cada operación y esto nos permite dividirlo en procesos más pequeños

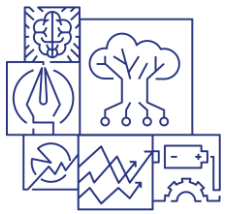


fidélitas Universidad

```
print("Bienvenido al programa")
sumar()
restar()
multiplicar()
dividir()
```

Desde el programa principal se hace el llamado a sub programas o procedimientos específicos para cada operación

El programa en general cuenta con 5 procesos, el principal y 1 por cada operación



fidélitas Universidad

```python
def sumar():
    a = int(input("Primer valor"))
    b = int(input("Segundo valor"))
    resultado = a + b
    print("La suma es: ", resultado)

def restar():
    a = int(input("Primer valor"))
    b = int(input("Segundo valor"))
    resultado = a - b
    print("La resta es: ", resultado)

def multiplicar():
    a = int(input("Primer valor"))
    b = int(input("Segundo valor"))
    resultado = a * b
    print("La multiplicación es: ", resultado)

def dividir():
    a = int(input("Primer valor"))
    b = int(input("Segundo valor"))
    resultado = a / b
    print("La división es: ", resultado)
```

Cada uno de los procesos se define de la siguiente manera

Pero esto tiene un inconveniente, cada vez que se ejecuta un proceso se van a solicitar los valores al usuario y esto no es muy funcional.

fidélitas Universidad


¿Qué son los parámetros?
¿Cómo utilizamos los parámetros?

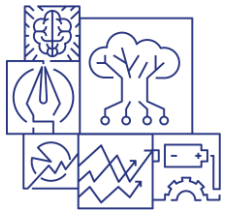
Primer Ejemplo

```
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
sumar(a,b)
restar(a,b)
multiplicar(a,b)
dividir(a,b)
```

Los valores solicitados en el proceso inicial

Una vez que hemos pedido los valores, los compartimos como parámetros por medio de los paréntesis en el llamado.

Se puede enviar los mismos valores en varios llamados.



fidélitas Universidad

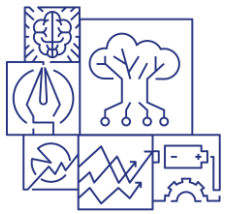
```
def sumar(x, y):
    resultado = x + y
    print("La suma es: ", resultado)

def restar(x, y):
    resultado = x - y
    print("La resta es: ", resultado)

def multiplicar(x, y):
    resultado = x * y
    print("La multiplicación es: ", resultado)

def dividir(x, y):
    resultado = x / y
    print("La división es: ", resultado)
```

En la definición del proceso se indican los parámetros que recibe.

En nombre del parámetro en el proceso principal no debe (pero puede) coincidir con el nombre dentro del proceso llamado. Lo importante es que sea la misma cantidad y del mismo tipo.

fidélitas Universidad

Debemos modificar cada procedimiento para que utilice la palabra reservada return junto con el valor a retornar.

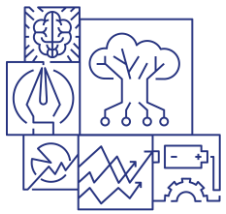


fidélitas Universidad

```
def sumar(x, y):
    resultado = x + y
    return resultado
```

Valores que recibe el proceso.

Valores que se retorna al punto de donde fue llamado



fidélitas Universidad

```
def sumar(x, y):
    resultado = x + y
    return resultado

def restar(x, y):
    return x - y

def multiplicar(x, y):
    resultado = x * y
    return resultado

def dividir(x, y):
    resultado = x / y
    return resultado
```

Todos los procesos tiene su instrucción return si se desea que retorne un valor

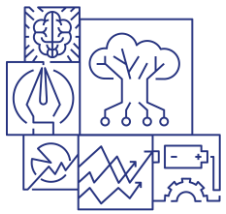
En la misma clausula return se puede realizar una operación aritmética

fidélitas Universidad

```
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
resultado = sumar(a,b)
print("La suma es: ", resultado)
print("La resta es: ", restar(a,b))
print("La multiplicación es: ", multiplicar(a,b))
print("La división es: ", dividir(a,b))
```

El valor que se retorna puede almacenarse en una variable para su posterior gestión

El proceso se puede utilizar en una concatenación u operación aritmética por la condición del mismo al retornar un valor



fidélitas Universidad

El programa completo

```
def sumar(x, y):
    resultado = x + y
    return resultado

def restar(x, y):
    return x - y

def multiplicar(x, y):
    resultado = x * y
    return resultado

def dividir(x, y):
    resultado = x / y
    return resultado

a = int(input("Primer número: "))
b = int(input("Segundo número: "))
resultado = sumar(a,b)
print("La suma es: ", resultado)
print("La resta es: ", restar(a,b))
print("La multiplicación es: ", multiplicar(a,b))
print("La división es: ", dividir(a,b))
```

Los procesos

El proceso inicial

Pruebe cambiando el orden de los procesos y colocando el inicial sobre los procesos de las operaciones

fidélitas Universidad

Ejercicio #1

Desarrolle un programa que convierta un número a su respectivo valor en base binaria, octal y hexadecimal.

Cada una de las conversiones debe ser un proceso independiente.

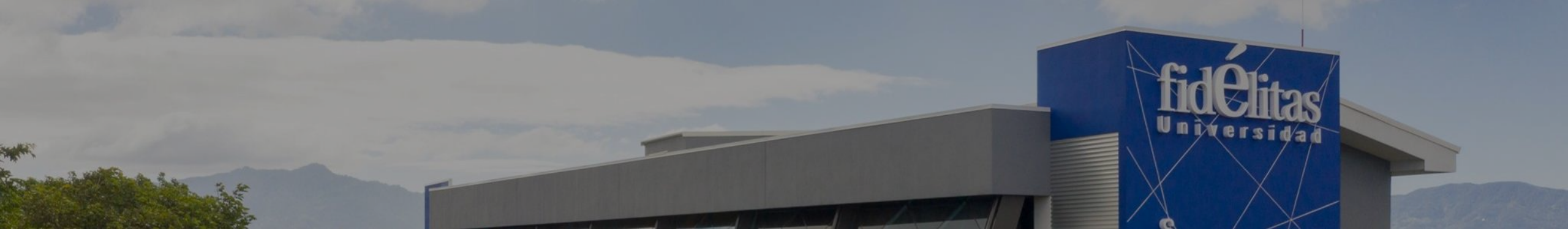
El proceso binario debe mostrar el resultado, los otros dos procesos deben retornar el resultado para ser mostrado desde el proceso inicial.

Adicionalmente, programe un proceso que reciba dos parámetros (el valor y la base) y que muestre el número correspondiente.

Tiempo aproximado 40 minutos

fidélitas Universidad

Con los sub programas tenemos la oportunidad de organizar nuestro código, mejorar la legibilidad y segmentar responsabilidades
.



¡Nos vemos la próxima semana!