

Programación

Clase 05

U2

Subprogramas



¡Importante recordar!

¿Qué sabemos sobre cómo programar hasta el momento?

Hasta el momento hemos visto lo siguiente:

- Tipos de datos básicos
 - Números enteros
 - Números decimales
 - Texto (Strings)
 - Booleanos (True or False)

¡Importante recordar!

- Condicionales

- if
- if – else
- if – elif – (else)

- Ciclos

- `for x in range(2):`
- `for x in range(2, 10):`
- `for x in range(2, 10, 2):`
- `while x >= 2:`
- `while linea != '':`
- `while True:`

Objetivo

Comprender el uso de subprogramas

Comprender los diferentes tipos de subprogramas y entender su uso

Comprender el alcance de los parámetros

Aplicar subprogramas para la resolución de problemas

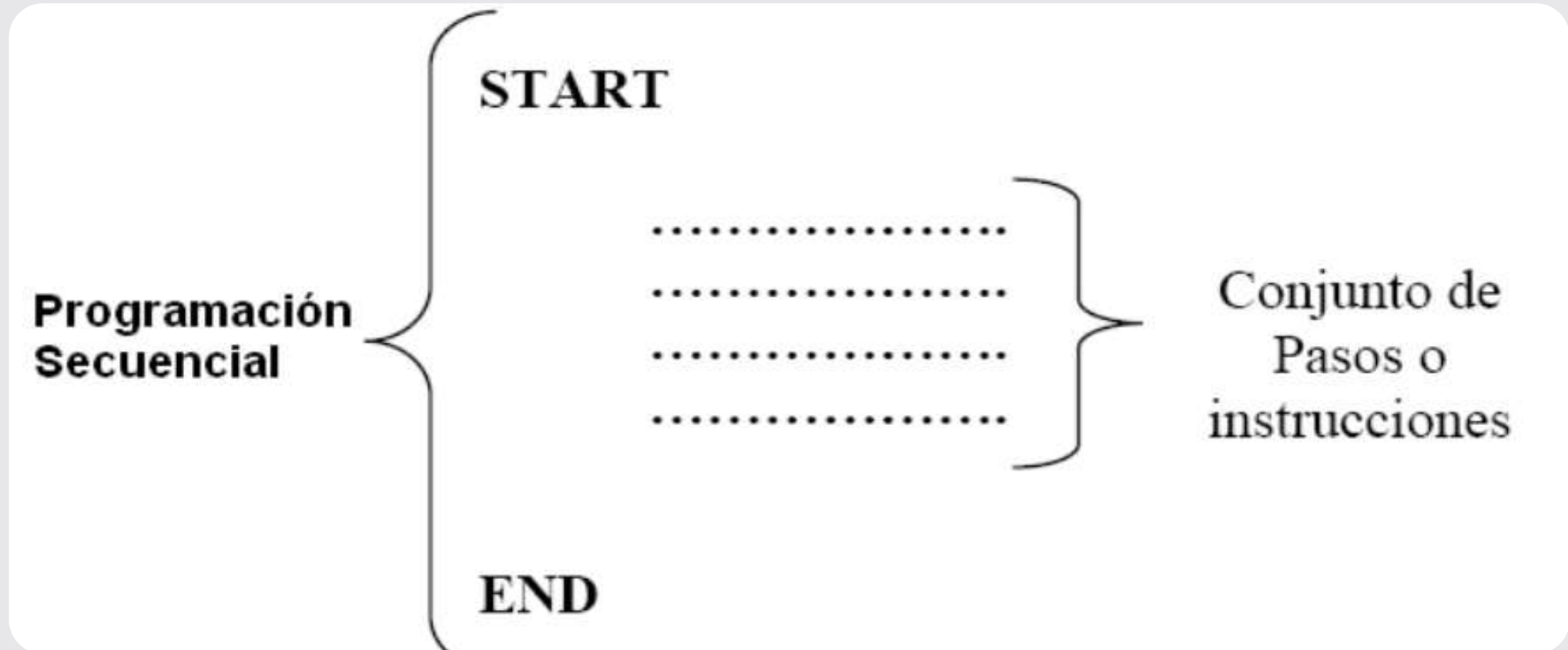
Introducción

Al resolver un problema complicado de programación, es necesario utilizar la técnica “**dividir para conquistar**”

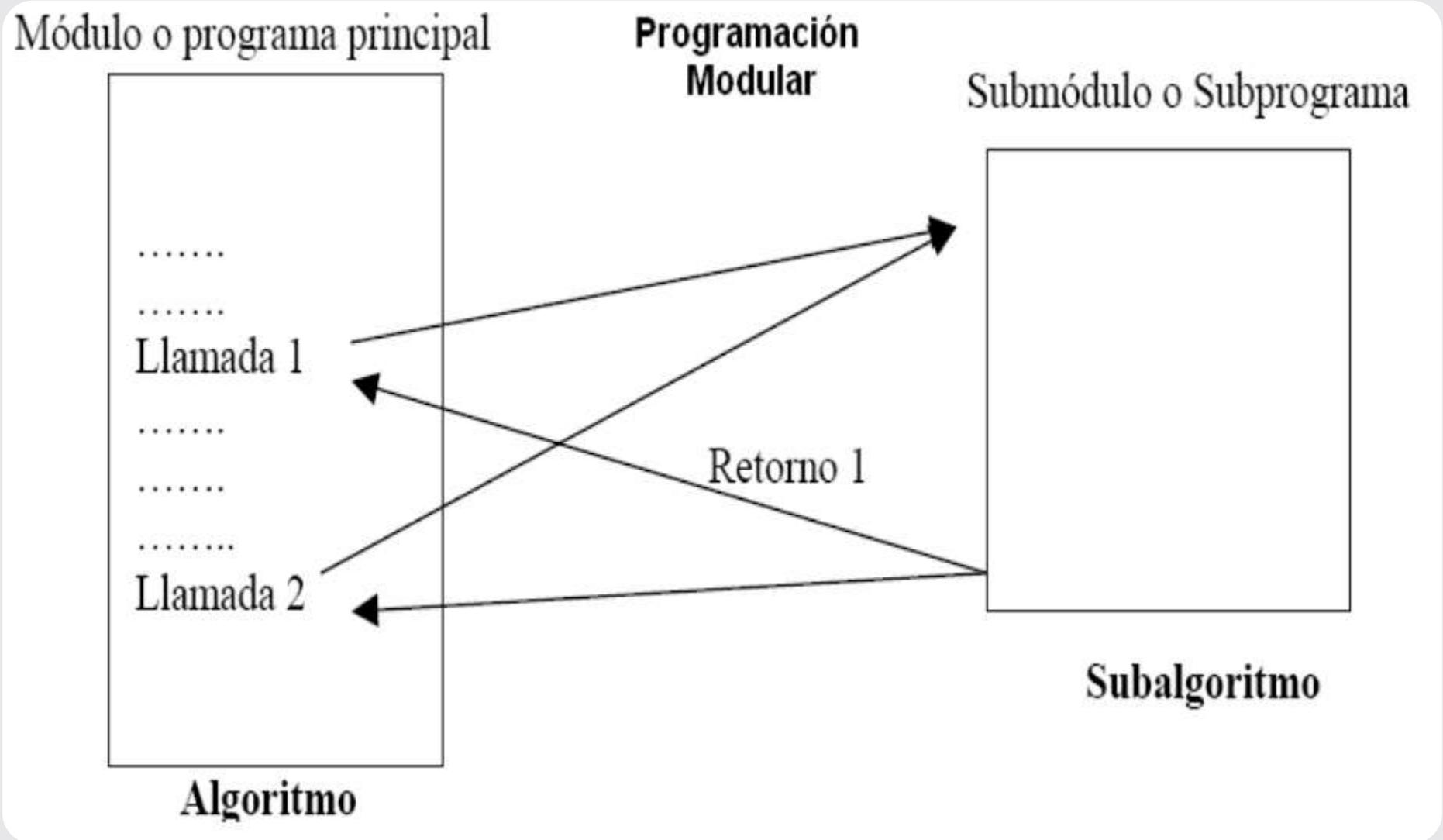
Es decir, si un problema es complejo, lo puedo dividir en subproblemas, cada uno de **menor complejidad**

Cada uno de estos subproblemas, lo puedo seguir dividiendo, de tal manera de llegar a problemas **sencillos** de resolver

O sea, vamos a pasar de esto...



A esto...



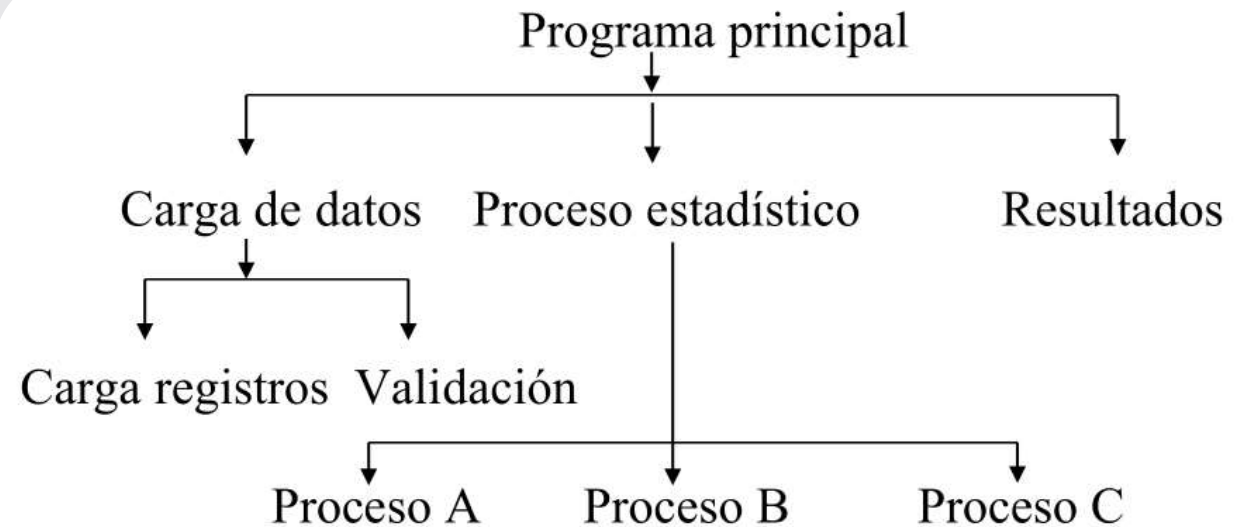
Ventajas de programar con subprogramas

Una gran ventaja al usar subprogramas es que permiten al programador diseñar el programa de tal forma que el cuerpo principal actúa como un supervisor, delegando tareas

El enfoque modular facilita la escritura y depuración de un programa, ya que las diferentes partes del programa (es decir, los subprogramas) pueden ser escritos y depurados independientemente

Beneficios

Un programa queda compuesto por un algoritmo coordinador, llamado programa principal, y un conjunto de subprogramas, llamados rutinas o subrutinas

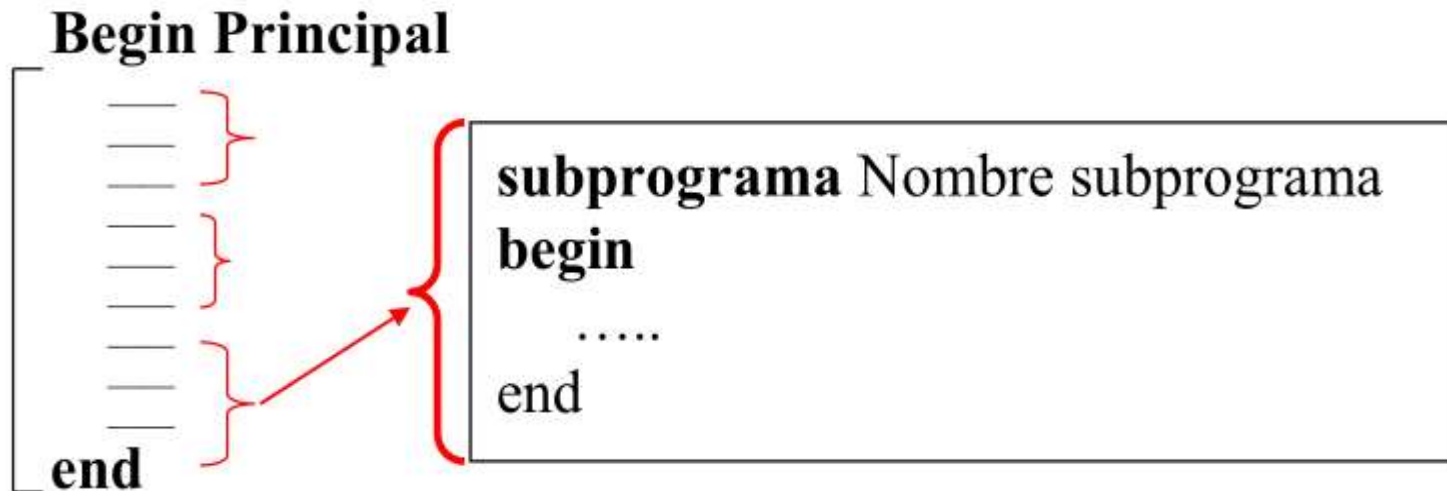


Beneficios

La coordinación entre el programa principal y las rutinas se logra por el algoritmo del programa y mediante la utilización de parámetros

Criterios para descomponer en subprogramas

1. Si existe un grupo de sentencias que se repite muchas veces, se puede formar un subprograma con estas sentencias, de tal manera que en cada lugar donde iban estas sentencias, se cambie por un llamado al subprograma



Begin Principal

Llamado al subprograma
====

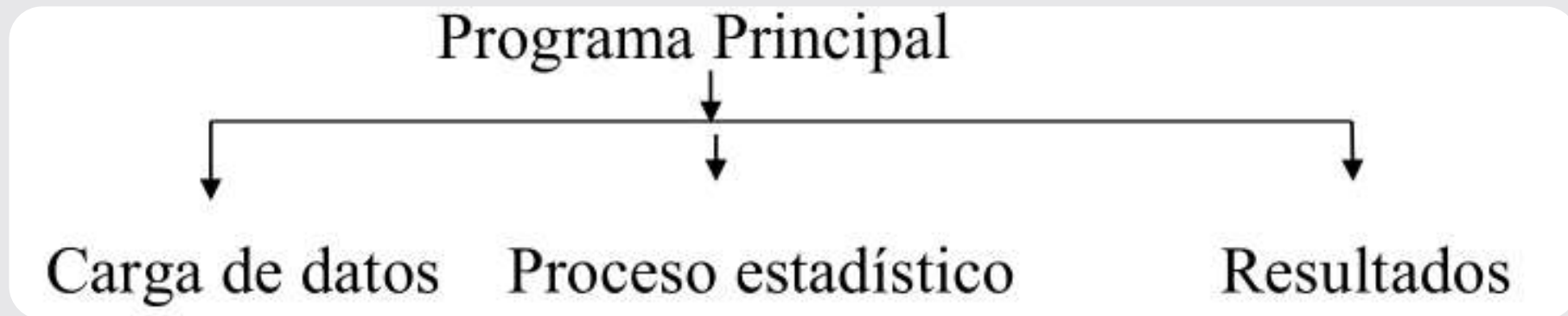
Llamado al subprograma

Llamado al subprograma

End

Criterios para descomponer en subprogramas

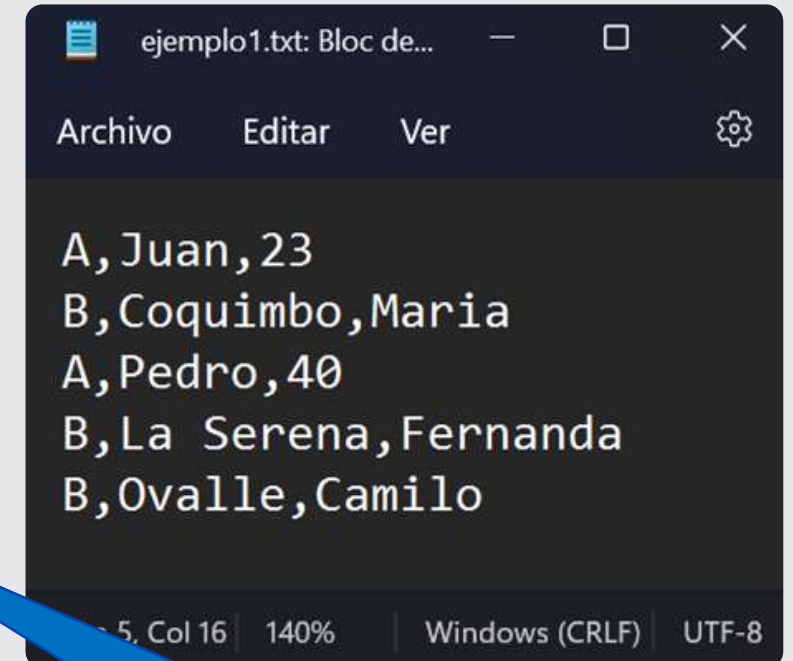
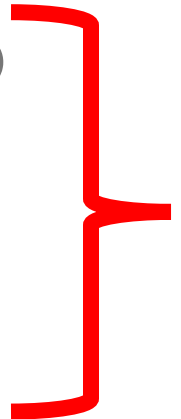
2. Dividir la funcionalidad del programa en partes más sencillas, donde cada parte tenga una funcionalidad clara y específica



Un ejemplo

```
arch = open('ejemplo1.txt', 'r')

linea = arch.readline().strip()
while linea != '':
    partes = linea.split(',')
    tipo = partes[0]
    if tipo == 'A':
        nombre = partes[1]
    else:
        nombre = partes[2]
    linea = arch.readline().strip()
```



**¡Lo podemos
hacer mejor!**

Pero, ¿qué tiene de malo?

¿Qué pasaría si hay otros tipos, en vez de solo A y B?

Tendríamos un **if** muy grande, ocupando mucho espacio y sin que sea la parte fundamental del código

El extraer el nombre ocupa mucho espacio, siendo que no es lo principal del algoritmo

```
arch = open('ejemplo1.txt', 'r')

linea = arch.readline().strip()
while linea != '':
    partes = linea.split(',')
    tipo = partes[0]
    if tipo == 'A':
        nombre = partes[1]
    else:
        nombre = partes[2]
```

Es más elegante escribir lo siguiente >>>

Versión 2

Aun cuando parece que es más código, estamos usando **dividir para conquistar**, y convirtiendo un problema complicado en uno **simple**

No te preocupes si no todo te hace sentido todavía. Solo mira la parte del while: ahora **es mucho más sencilla**

```
arch = open('ejemplo1.txt', 'r')
```

```
def obtenerNombre(linea):  
    partes = linea.split(',')  
    tipo = partes[0]  
    if tipo == 'A':  
        nombre = partes[1]  
    else:  
        nombre = partes[2]  
    return nombre
```

```
linea = arch.readline().strip()  
while linea != '':  
    nombre = obtenerNombre(linea)  
    linea = arch.readline().strip()
```

Sintaxis en Python

The diagram illustrates the syntax of a Python function definition. It shows a function named `multiplicar` that takes two parameters, `param1` and `param2`. The function body contains several actions: `acción 1`, `acción 2`, an ellipsis (`...`), and `acción n`. These actions are grouped by a blue box, with an arrow pointing to the label **Acción**. The function concludes with a `return` statement followed by `resultado`, which is underlined with a blue line and has an arrow pointing to the label **Salida (opcional)**. The word **Entrada** is positioned above the parameters, with two arrows pointing to `param1` and `param2` respectively.

```
def multiplicar(param1, param2):  
    acción 1  
    acción 2  
    ...  
    acción n  
    return resultado
```

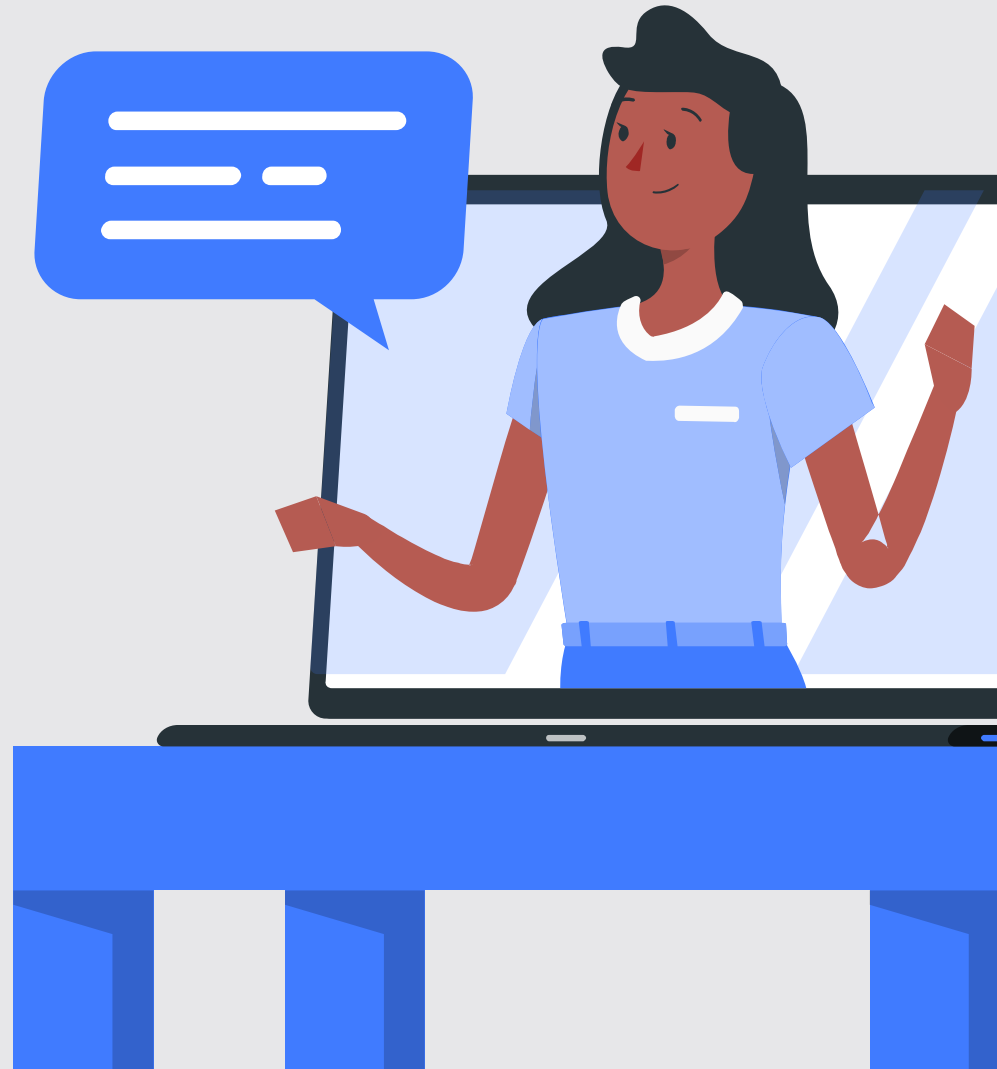
Entrada

Acción

Salida (opcional)

Clase 06: Subprogramas

Tipos de subprogramas



Dos tipos

De acuerdo a cómo se usan, los subprogramas se pueden clasificar en dos tipos:

Funciones

Procedimientos

Hay procedimientos que ya conoces

```
print('Hola, ¿cómo estás?')
```

¿Qué hace este procedimiento? (o sea, cuál es su comportamiento)

¿Cómo crees que funciona?

¿Cómo sabe el procedimiento **qué** es lo que quieres imprimir por pantalla?

Hay funciones que ya conoces

```
arch = open('archivo.txt', 'r')
```

¿Qué hace esta función? (o sea, cuál es su **comportamiento**)

¿Cómo crees que funciona?

¿Cómo sabe la función **cuál** es el archivo que quieres abrir para leer?

Procedimiento vs Función

Las funciones son subprogramas que después de procesadas sus instrucciones, **retornan** un valor

Imagina algo como una función matemática:

$$f(x) = 3x^2 + 2x - 1$$

Al invocar la función, ésta nos entrega un valor

Procedimiento vs Función

Los **procedimientos** son subprogramas que después de procesadas sus instrucciones, **no retornan** un valor

Modificando el comportamiento de subprogramas

La forma normal de **modificar** el comportamiento de un subprograma es a través de sus parámetros.

Los **parámetros** permiten **comunicar valores** al subprograma, que puede usar de la forma en que quiera:


- Extraer datos
- Realizar cálculos
- O hasta ignorarlos (no es una buena práctica)

En el caso del **print**, el parámetro le indica lo que se quiere imprimir por pantalla

Parámetros

Desde el punto de vista del subprograma, los parámetros se comportan como variables normales

Pero, al momento de escribir la función o procedimiento, hay que especificar dichos parámetros, y cómo se llamarán:



```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')
```


Parámetros

Recuerda que cuando se definen parámetros se

La cantidad de parámetros a definir es ilimitada

Y un subprograma se puede usar desde dentro de otro subprograma (fíjate que estamos usando print dentro de escribirElMenor)

```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')
```

¿Y cómo se usa un subprograma?

Un subprograma definido, pero nunca usado, es solamente **código muerto**

Todo el motivo de crear un subprograma es para poder usarlo

A partir de ahora, al hecho de usar un subprograma lo llamaremos **llamar** o **invocar** al subprograma

Cuando **llamamos** a un subprograma, estamos transfiriendo el flujo normal del programa al subprograma, y éste continuará una vez que el subprograma complete su ejecución

Flujo del programa

```
# Programa principal
```

```
...
```

```
...
```

```
...
```

```
escribirElMenor(i, j)
```

```
...
```

```
...
```

```
...
```

```
escribirElMenor(a, b)
```

```
...
```

```
...
```

```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')
```

Invocando procedimientos

Para llamar a un procedimiento, simplemente basta escribir su nombre, y especificar el valor de sus parámetros.

```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')  
  
escribirElMenor(10, 20)  
escribirElMenor(100, 2)  
  
for x in range(10):  
    escribirElMenor(x, 20)
```

Invocando funciones

Para llamar a un función, además de escribir su nombre, y especificar el valor de sus parámetros, necesitamos guardar el valor de retorno

Fíjate que dentro de la función, usamos la palabra clave **return**

```
def determinarElMayor(num1, num2):  
    mayor = num2  
    if num1 > num2:  
        mayor = num1  
    return mayor  
  
m = determinarElMayor(50, 10)  
print(m)  
  
for x in range(1, 11):  
    print(determinarElMayor(5,x))
```

Un pequeño ejemplo

```
def calcularBono(s):  
    b = 0  
    if s > 10000:  
        b = 100  
    if s < 5000:  
        b = 200  
    return b
```

```
def imprimir(b, mensaje):  
    print('Estimado trabajador')  
    print(mensaje)  
    print(b)
```

```
sueldo = int(input('Ingresa tu sueldo: '))  
bono = calcularBono(sueldo)  
imprimir(bono, 'El valor de tu bono es')
```

¿Cuándo usar una función o un procedimiento?

Se usan funciones cuando se requiere un resultado concreto:

La función entrega un resultado, que reemplaza la invocación de ésta en el código:

```
m = 3.14 + sqrt(x)
```

```
if valorAbsoluto(x) > 10:  
    print(x)
```

```
while obtenerNombre(linea) != 'FIN':
```

¿Cuándo usar una función o un procedimiento?

Los procedimientos se usan cuando no se requiere un valor de retorno, pero sí se quiere tener instrucciones que se pueden reutilizar y ser llamadas más de una vez

```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')  
  
escribirElMenor(10, 20)  
escribirElMenor(100, 2)  
  
for x in range(10):  
    escribirElMenor(x, 20)
```


Parámetros

Como ya hemos visto, los subprogramas reciben **parámetros** para **alterar** su comportamiento

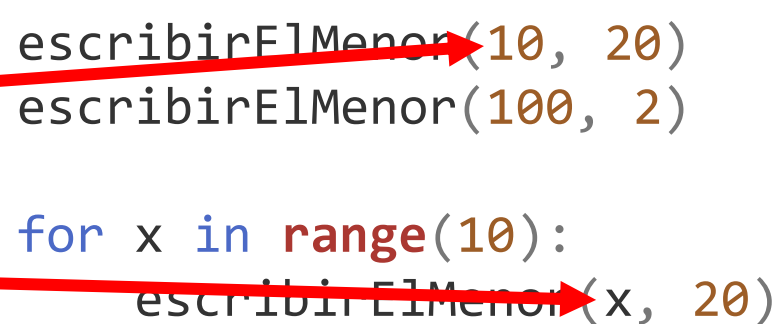
Desde el punto de vista del subprograma, los parámetros son **variables normales**, y pueden ser usadas como tales.

Desde el punto de vista del que **invoca** al subprograma, los valores que se pasan a la función son los valores del elemento en el momento en que fue invocado el subprograma

Parámetros

Al invocar el procedimiento, los parámetros adquieren los valores especificados al hacer la llamada (ya sean valores literales, o valores en variables)

```
def escribirElMenor(numero1, numero2):  
    if numero1 < numero2:  
        print('El menor es', numero1)  
    elif numero2 < numero1:  
        print('El menor es', numero2)  
    else:  
        print('Los dos números son iguales')  
  
escribirElMenor(10, 20)  
escribirElMenor(100, 2)  
  
for x in range(10):  
    escribirElMenor(x, 20)
```



Ejemplo de la vida real

Le compraste un chocolate a tu madre para el día de la madre y necesitas envolverlo en papel de regalo.

Si lo vas a envolver tú, entonces tú podrías ser el subprograma cuya **acción** es **ENVOLVER_REGALO**

¿Qué necesitarías para envolver el regalo?

Exacto, necesitarías **papel de regalo**.

Entonces tu **parámetro de entrada** sería el papel de regalo, tu **acción** sería envolverlo, y tu **salida** sería el regalo envuelto.

Ejemplo en programación

Necesitamos multiplicar dos números

¿Qué necesitaríamos para multiplicar los dos números?

Exacto, el **número1** y el **número2**

¿Cuál sería la acción?

Sí, la acción sería **multiplicar**

¿Cuál sería la salida?

Exacto, el **resultado** de la multiplicación

Ejemplo en programación

Si transformáramos lo anterior a código en Python sería de la siguiente manera:


```
def multiplicar(numero1, numero2):  
    resultado = numero1 * numero2  
    return resultado
```

The diagram illustrates the components of the Python function `multiplicar`. The word **Entrada** (Input) is positioned above the function parameters `numero1` and `numero2`, with two green arrows pointing down to them. The assignment statement `resultado = numero1 * numero2` is enclosed in a blue rectangular box. The word **Acción** (Action) is positioned to the right of this box, with a green arrow pointing from the box to it. The `return` statement is underlined in blue. The word **Salida** (Output) is positioned below the underlined `return` statement, with a green arrow pointing from the underlined text to it.

Ejemplo en programación

¿Este subprograma es un procedimiento o función?

```
def multiplicar(numero1, numero2):  
    resultado = numero1 * numero2  
    return resultado
```



Sí, es una **función** porque **retorna** un valor

Ejemplo en programación

¿Este subprograma es un procedimiento o función?

```
def suma(numero1, numero2):  
    suma = numero1 + numero2  
    print(suma)
```

No es una función. Es un **procedimiento** porque **NO** retorna nada

Paso de variables

En este punto es conveniente que aprendas acerca de algo que tiene que ver con las variables, y que te será muy útil al usar funciones y procedimientos.

Analiza este código

```
def intercambiador(num1, num2):  
    num1 = num2  
    num2 = num1  
  
a = 10  
b = 20  
  
print(a, b)  
intercambiador(a, b)  
print(a, b)
```

10 20
10 20

¿Qué pasó?

Aún cuando pareciera que el procedimiento hace algo con los parámetros que recibe, esos cambios son **LOCALES** al procedimiento.

El valor de las variables **a** y **b** se **COPIA** a los parámetros.

Cualquier cambio no se **“ve”** afuera.

A esto se le llama **“pasar por valor”** las variables.

```
def intercambiador(num1, num2):  
    num1 = num2  
    num2 = num1
```

```
a = 10  
b = 20
```

```
print(a, b)
```



OJO que esta NO es la forma correcta de intercambiar el valor de dos variables
(esto se puso acá solo como ejemplo)

Alcance

Analiza este código

```
def sumador(num1, num2):  
    suma = num1  
    suma += num2  
  
a = 10  
b = 20  
  
sumador(a, b)  
print(suma)
```


NameError: name 'suma' is not defined

Alcance

Fíjate que en la última línea se está creando la variable **suma**.
La variable **suma** fue creada y solo es visible dentro del procedimiento

Pregúntate: ¿dónde está declarada la variable suma?

La variable **suma** literalmente no es visible en este contexto



```
def sumador(num1, num2):  
    suma = num1  
    suma += num2  
  
a = 10  
b = 20  
  
sumador(a, b)  
print(suma)
```

Alcance

Las variables que se crean dentro de un procedimiento o función solamente son visibles dentro de dicho contexto

Esta variable **NO EXISTE**



```
def imprimir(mensaje, numero):  
    texto = 'Hola '  
    texto += mensaje  
    texto += ' el número es '  
    texto += str(numero)  
    return texto
```

```
print(imprimir('Ximena', 101))  
print(texto)
```

NameError: name 'texto' is not defined

Alcance

¿Qué crees que va a pasar en el último

No 'entra' a ninguna condición

Por lo que al llegar acá, 'precio' no existe

Si tipo es 'error'

```
def ventas(tipo, cantidad):  
    if tipo == 'cancha':  
        precio = 20000  
    elif tipo == 'platea':  
        precio = 30000  
    elif tipo == 'vip':  
        precio = 40000  
    total = cantidad * precio  
    return total
```

```
print(ventas('vip', 4)) # 160000  
print(ventas('error', 1)) # ?
```

UnboundLocalError: local variable 'precio' referenced before assignment

¿Por qué no había tenido este problema antes de empezar a usar funciones?

Cuando no usamos funciones ni procedimientos, todas las variables se crean en un contexto que se llama **“alcance global”**

Básicamente todo es visible!

```
suma = 0
a = 1
b = 1

while a < 10:
    c = a + b
    suma += c

    a = b
    b = c

print(suma)
```

Aunque también pueden suceder problemas similares

NameError: name 'especial' is not defined

```
suma = 0
a = 1
b = 1

while a < 10:
    c = a + b
    suma += c

    if a > 1000:
        especial = 1000

    a = b
    b = c

print(especial)
```

Resumiendo

¡SUBPROGRAMAS!

¿Qué es un subprograma?

Respuesta: un subprograma es un bloque de código que realiza una acción específica

¿Cuántos tipos de subprogramas tenemos?

Respuesta: funciones y procedimientos

Procedimiento vs. Función

Procedimiento

Puede o no recibir **parámetros**

No retorna nada

Función

Puede o no recibir **parámetros**

Sí retorna un valor

Trabajo autónomo

Revisar capítulo 8 libro guía.

Resolver ejercicios 1, 2, 5, 6, 11, 12, 13, 14