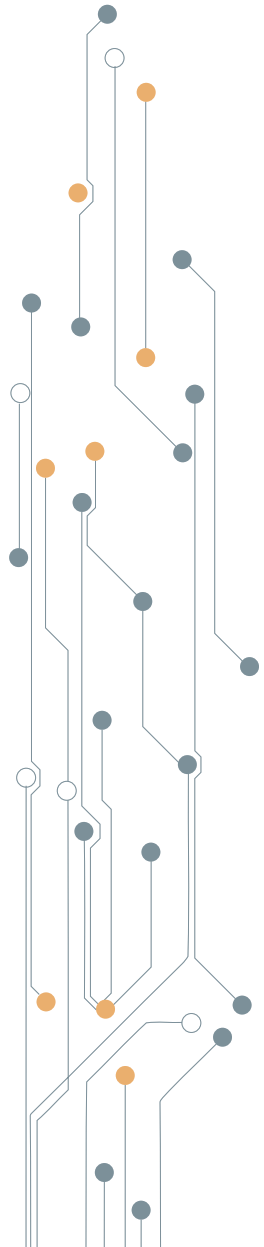
Programación Modular

Telefónica
EDUCACIÓN DIGITAL

Índice



Programación Modular

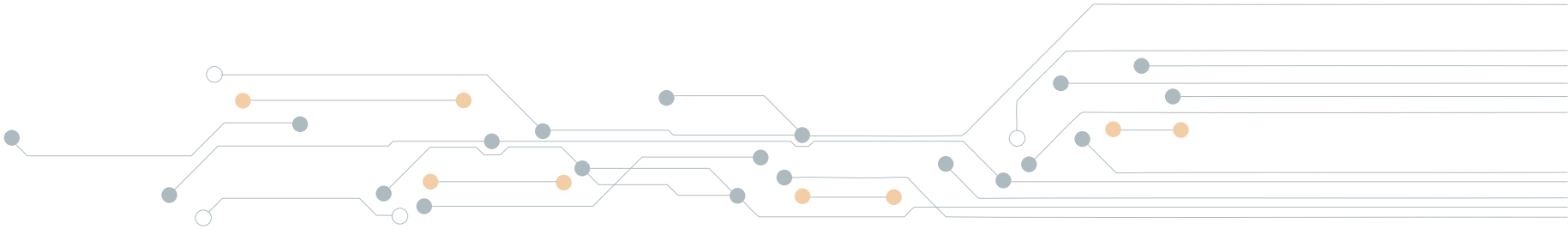
1. Programación Modular

La programación modular representa la evolución natural de la programación estructurada. Basada en la idea de "divide y vencerás", la programación modular consiste en dividir un programa grande en bloques más pequeños (módulos), a fin de que el desarrollo pueda acometerse con mayor facilidad.

1.1 | Teorema de la estructura

Son numerosas las ventajas que obtenemos al dividir un programa grande en bloques más pequeños, entre ellas destacan:

- **Simplificación del desarrollo.** Al dividir un problema grande en problemas más pequeños, podemos concentrarnos en acometer esos problemas pequeños de manera independiente, facilitando su resolución.
- **Menor número de errores.** Dividiendo el desarrollo en programas más simples, la probabilidad de cometer errores en estos mini programas es menor que en un programa grande. Además, los errores cometidos son más fáciles de detectar y, por tanto, de resolver.



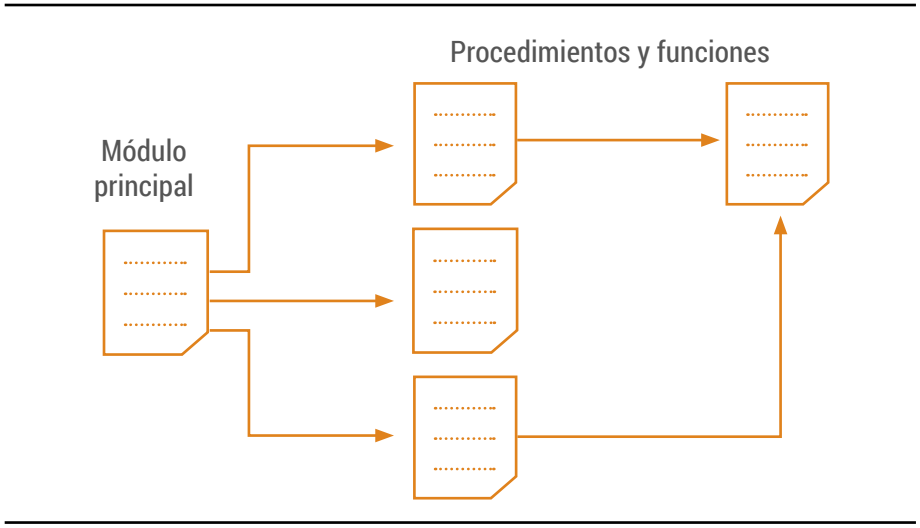
- **Reutilización de código.** Se trata de una ventaja muy interesante, pues al disponer de módulos encargados de realizar una determinada tarea, estos módulos pueden utilizarse en distintas partes del programa donde se requiera realizar dicha tarea, evitando tener que reescribir las instrucciones de nuevo

- **Separación entre capas.** Con la programación modular se facilita la separación entre capas de la aplicación, es decir, que cada capa se pueda programar por separado. Le división de una aplicación en capas permite dividir la aplicación en funcionalidades, por ejemplo, la capa de presentación se encargaría de todo lo relativo a entrada y salida de datos, mientras que la capa de lógica de aplicación se encargaría de procesar los datos y realizar los cálculos con los mismos.

1.2 | Procedimientos y funciones

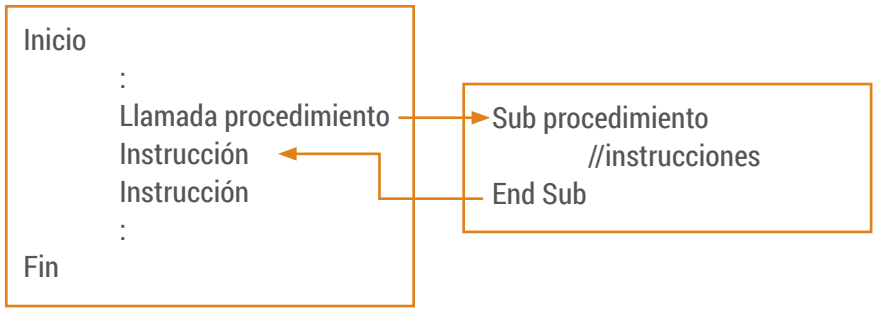
La manera que tenemos en programación de aplicar la modularidad es a través de los **procedimientos y las funciones.** Ambos representan bloques de código que realizan alguna tarea y que pueden ser llamados desde otra parte del programa.

Por ejemplo, las diferentes tareas a realizar pueden ser implementadas en procedimientos y funciones y desde un módulo principal hacer llamadas a los mismos cuando se requiera ejecutar la tarea que llevan asociada:



PROCEDIMIENTOS

Un procedimiento es un bloque de código que se ejecuta cuando es llamado desde otra parte del programa. Tras finalizar su ejecución, el control de la ejecución se pasa a la línea siguiente a la que provocó la llamada.



En notación de pseudocódigo, un procedimiento lo definiremos de la siguiente manera:

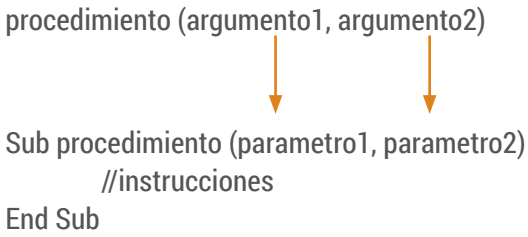
```
Sub nombre_procedimiento(parametro1, parametro2, …)
        //instrucciones
End Sub
```

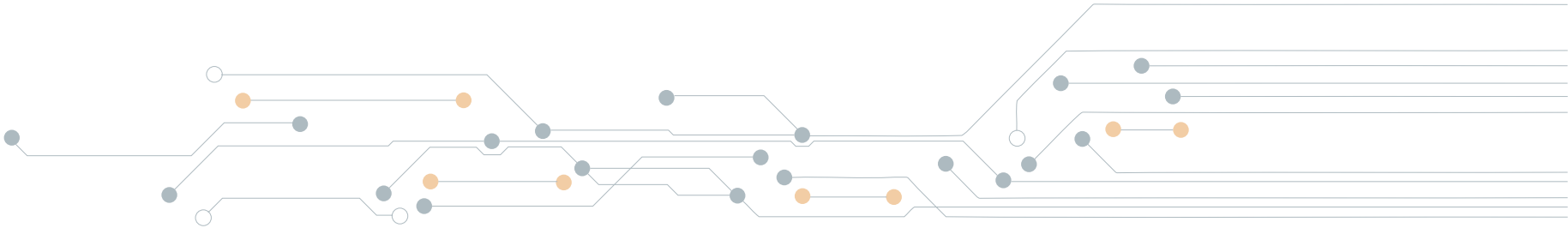
La palabra *Sub* indica el inicio del procedimiento, a continuación se indica el nombre del mismo y entre paréntesis los parámetros, que no son más que las variables donde se volcarán los datos que el procedimiento recibirá en la llamada. De cara a indicar los parámetros, se especificará también el tipo de datos que representan.

Para llamar a un procedimiento desde otra parte del programa utilizaremos el nombre del mismo, seguido de la lista de argumentos o valores de llamada entre paréntesis:

```
nombre_procedimiento (argumento1, argumento2,…)
```

Los argumentos son los datos que se pasan en la llamada al procedimiento, los cuales **se volcarán en los parámetros** indicados en el mismo.





El argumento utilizado en la llamada a un procedimiento puede ser una variable o un literal.

Veamos un ejemplo. A continuación, definimos un sencillo procedimiento cuya misión es mostrar el mayor de los dos números recibidos como parámetros:

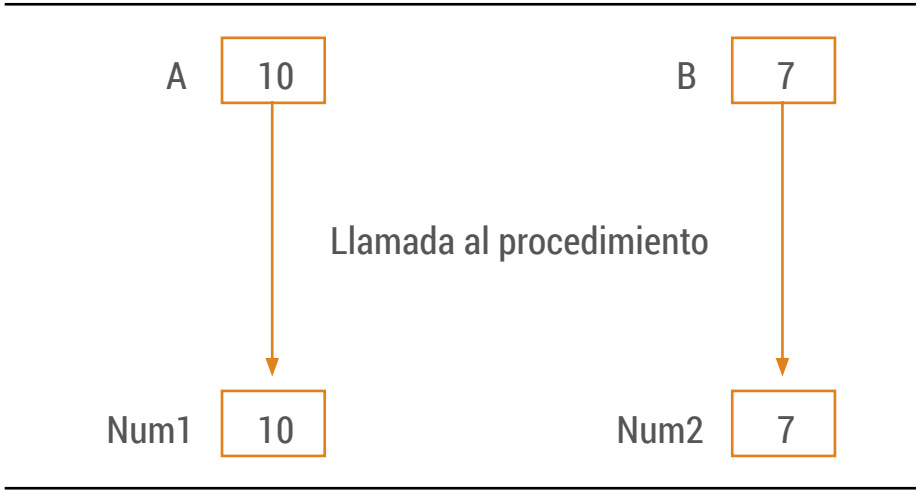
```
Sub MostrarMayor(num1 Integer, num2 Integer)
      If(num1>num2) Then
            Mostrar "El mayor es ", num1
      Else
            If((num2>num1) Then
                  Mostrar "El mayor es ", num2
            Else
                  Mostrar "Los números son iguales"
            End If
      End If
End Sub
```

Como vemos en el ejemplo anterior, la declaración de los parámetros de un procedimiento se realiza igual que la declaración de variables en un programa, solo que al declararse entre los paréntesis del procedimiento, **solo pueden utilizarse en el interior de éste.**

Si desde otra parte del programa se leen dos números y se quiere indicar cuál de los dos es el mayor, utilizando el procedimiento descrito anteriormente, el pseudocódigo sería simplemente:

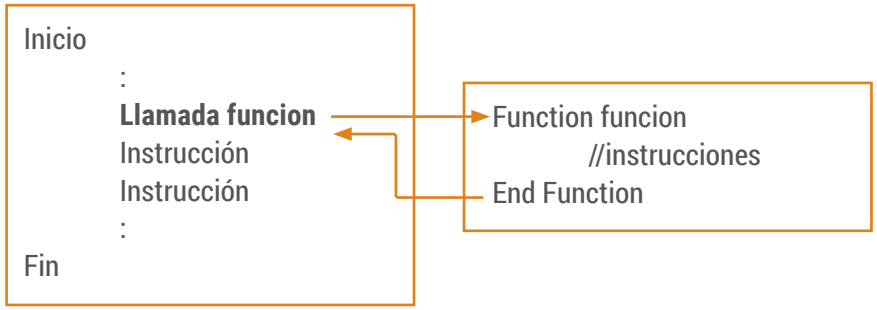
```
Leer A, B
MostrarMayor(A, B)
```

En este caso se pasan como argumentos en la llamada al procedimiento los valores contenidos en las variables A y B. Estos se volcarán en las variables parámetro num1 y num2, que son con las que trabajará el procedimiento.



FUNCIONES

Al igual que un procedimiento, una función es un bloque de código que se ejecuta cuando es llamada y puede recibir también unos parámetros de entrada, pero a diferencia de aquel, devuelve un resultado al punto de llamada:



Para definir una función en pseudocódigo utilizaremos la palabra **Function** en vez de Sub

```
Function nombre_funcion (parametro1, parametro2, …)
      //instrucciones
      Return resultado
End Function
```

La instrucción *Return* se utiliza para devolver el resultado al punto de llamada, donde se recogerá y será procesado de alguna manera.

La instrucción de llamada seguirá el siguiente formato:

```
variable=nombre_funcion(argumento1, argumento2,..)
```

Cuando un programa ejecute esta instrucción, se realizará una llamada a la función indicada y se le pasarán los argumentos especificados entre paréntesis. El control del programa pasará entonces a la función, que tras ejecutar el Return, devolverá el control al punto de llamada y el resultado se volcará en la variable.

Como ejemplo, presentamos una función encargada de devolver el mayor de dos números recibidos:

```
Function ObtenerMayor(num1 Integer, num2 Integer)
      //declara la variable donde guardará el resultado
Datos:
      resultado Integer
      Código:
      If(num1>num2) Then
            resultado=num1
      Else
            resultado=num2
      End If
Return resultado
End Function
```

Como hemos visto en el listado anterior, una función o procedimiento también tendrá su zona de declaración de variables y zona de código.

Utilizando la función anterior, el siguiente bloque de código se encargaría de leer dos números y mostrar un mensaje indicando cual es el mayor:

```
Leer N1, N2
res=ObtenerMayor(N1, N2)
Mostrar “El mayor de los números es:”, res
```

Se podría realizar la llamada a la función en la misma instrucción de salida sin necesidad de utilizar la variable *res*. Hay que tener en cuenta, que cuando una llamada a una función se realiza dentro de otra instrucción, se pasa el control a dicha función y cuando finaliza se completa la ejecución de la instrucción donde se encuentra la llamada a la función, utilizando el valor devuelto por esta:

```
Leer N1, N2
Mostrar “El mayor de los números es:”, ObtenerMayor(N1, N2)
```

2. Programación por capas

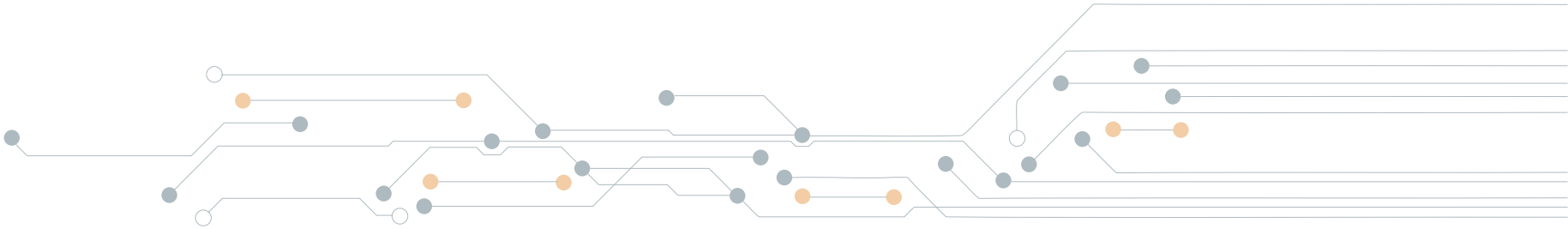
Como ya hemos indicado, la programación modular permite dividir un programa grande en pequeñas partes más fáciles de desarrollar.

A la hora de aplicar esta metodología en el diseño de algoritmos mediante pseudocódigo, lo que haremos será definir en una serie de procedimientos y funciones los algoritmos encargados de realizar las distintas tareas de procesamiento de datos del programa.

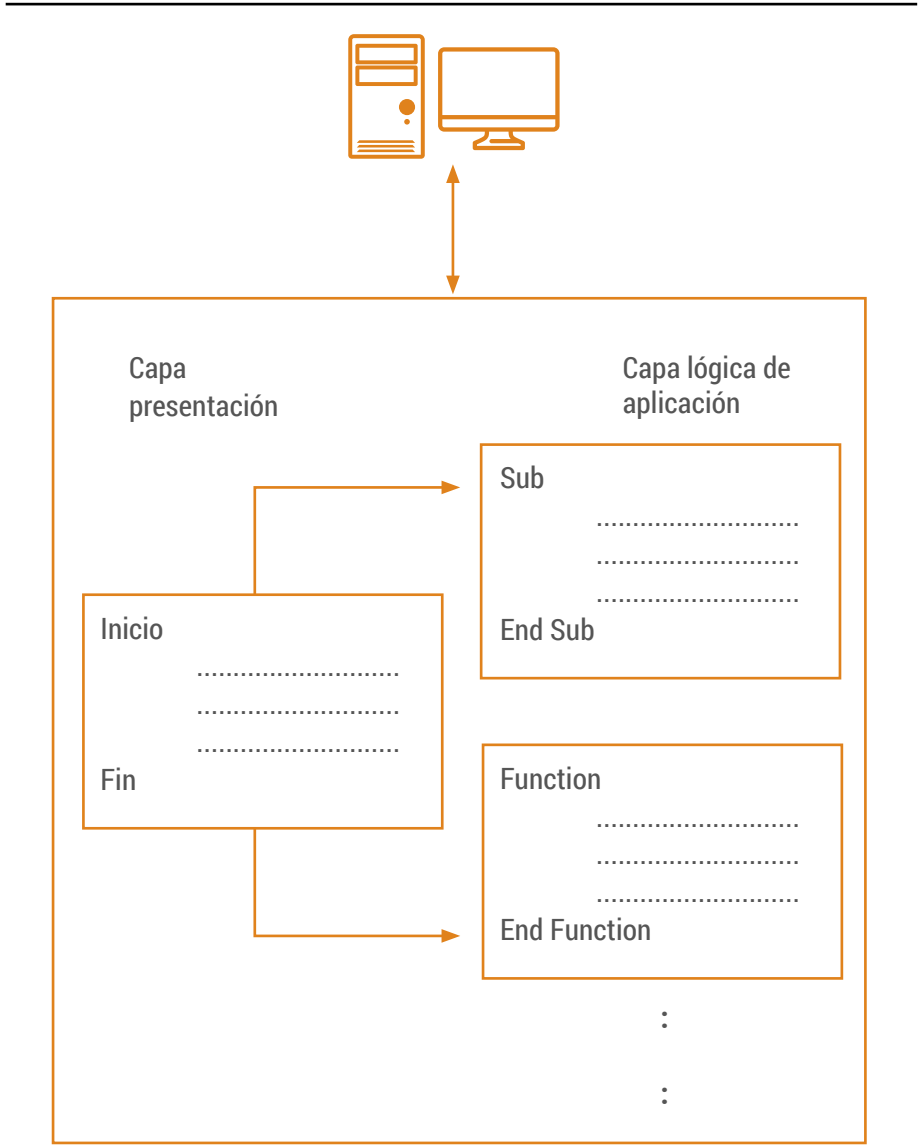
En el módulo principal, que es el que delimitaremos entre las etiquetas *Inicio* y *Fin*, nos encargaremos de la lectura de datos del exterior y la presentación de resultados. Desde éste llamaremos a los procedimientos y funciones creados para realizar el procesamiento de los datos.

Con esta forma de diseñar los programas, conseguimos separar el desarrollo del pseudocódigo (y también el del código final) en dos capas:

- **Capa de presentación.** Será la encargada de la interacción con el usuario, es decir, de solicitar datos al mismo y presentar los resultados obtenidos del procesamiento de dichos datos. El algoritmo correspondiente a esta capa se define dentro del módulo principal.



- **Capa de lógica de aplicación.** Realiza las operaciones con los datos y genera los resultados del procesamiento de los mismos. Se implementa a través de funciones y procedimientos, que son llamados desde el módulo principal. En esta capa no se codifican instrucciones de entrada y salida de datos; dichas operaciones son realizadas desde la capa de presentación.



Telefónica

EDUCACIÓN DIGITAL