Introducción a la Programación

Origen de la programación

Programación hace referencia al proceso mediante el cual se diseña y codifica un conjunto de instrucciones que implementan un determinado algoritmo.

Surge ante la necesidad de automatizar y facilitar tareas que se realizaban de forma manual (por ejemplo operaciones matemáticas elementales o instrucciones en máquinas de telares automáticos).

Telar de Jacquard

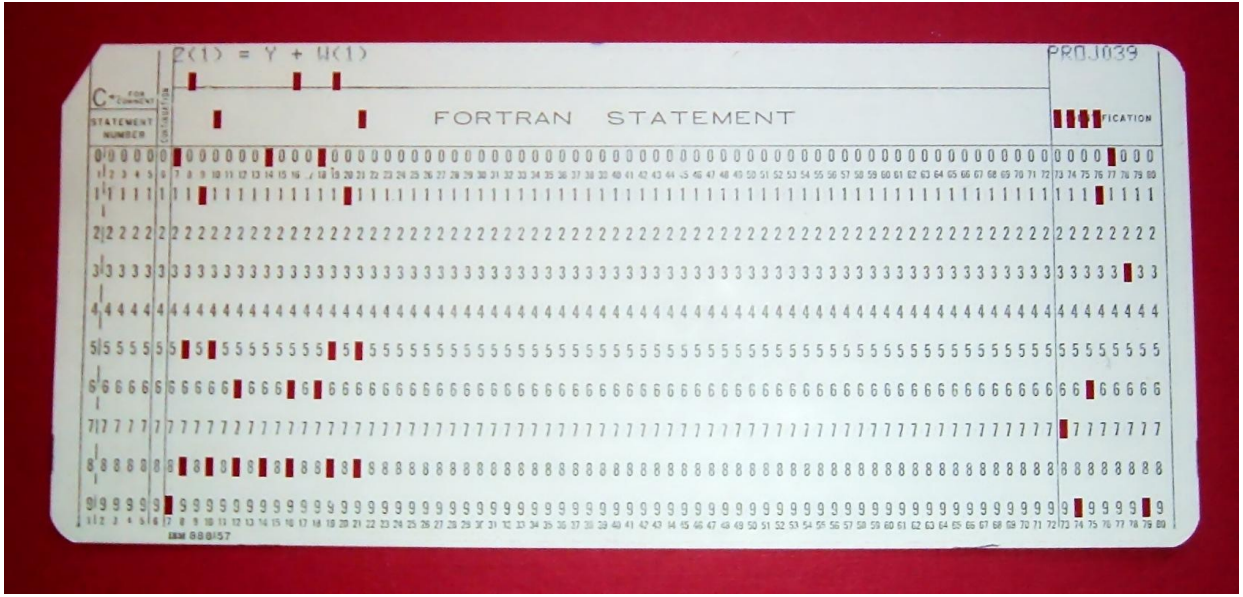
(Joseph Marie Jacquard en 1801)

Orígen de la programación

La mayor revolución surge de la aparición de la máquina analítica de *Charles Babbage* (considerado el primer ordenador de la historia) y la aportación de *Ada Lovelace* (considerada la primera programadora de la historia)

Orígen de la programación

La programación ha ido evolucionando desde las primeras instrucciones, escritas en código binario (ceros y unos) a nuevos modelos más fáciles de usar en los que se utilizaban palabras para realizar las instrucciones (lenguaje ensamblador) y posteriormente se evolucionó a los lenguajes de alto nivel.



Tarjeta perforada

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


 C000                    ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START    LDS    #STACK

                *************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                *************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                *************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

Lenguaje ensamblador

```
public class PDatosImpresion
{
    public string NombreEtiqueta { get; set; }
    public string NombreImpresora { get; set; }
    public int NumeroEtiquetas { get; set; }
    public string Talla { get; set; }
    public List<string> ListadoTallas { get; set; }
    public string CodigoSuperior { get; set; }
    public string CodigoInferior { get; set; }
    public string Precio { get; set; }

    public PDatosImpresion()
    {

    }

    public PDatosImpresion(string datosFichero)
    {

        string[] datosArray= (datosFichero.Split(';'));

        if(datosArray.Length<2)
            throw new EtiquetasJealferException("El número de campos del

        this.NombreEtiqueta = datosArray[0];
        this.NombreImpresora = datosArray[1];

        string nombreEtiquetaExterna = ConfigurationManager.AppSettings["
        string nombreEtiquetaInterna = ConfigurationManager.AppSettings["
        string nombreEtiquetaInterna2 = ConfigurationManager.AppSettings[
        string nombreEtiquetaInterna3 = ConfigurationManager.AppSettings[
        string nombreEtiquetaInterna4 = ConfigurationManager.AppSettings[

        if (NombreEtiqueta==nombreEtiquetaExterna)
        {
            this.ObtenerDatosEtiquetaExterna(datosArray);
        }
```

Lenguaje de alto nivel

Paradigmas de la programación

Los paradigmas de la programación son un conjunto de reglas que son aceptadas como modelo de referencia para realizar código de calidad.

Los paradigmas no son excluyentes y se suelen utilizar de forma conjunta para el desarrollo de programas, lo que se suele llamar *programación multiparadigma*.

En esta parte de introducción veremos dos paradigmas de programación:

1. Programación estructurada
2. Programación modular

Programación Estructurada

Deriva del *paradigma de la programación imperativa*, que indica cómo debe resolverse un problema ejecutando instrucciones que al ejecutarse varían el estado del programa.

Se basa en que todo programa se puede escribir usando tres estructuras de control:

- **Secuencial**: Instrucciones una detrás de otra siguiendo un orden.
- **Alternativa**: Se evalúan expresiones y, dependiendo de su resultado, se decide la siguiente instrucción a ejecutar.
- **Iterativa**: Repetición de instrucciones hasta cumplir una condición.

Programación Modular

El código, al crecer mucho, complica mucho el desarrollo cosa que no es tenida en cuenta por la programación estructurada.

La programación modular propone como solución la descomposición del problema en varios subproblemas de menor tamaño pudiendo estos ser descompuestos a su vez en otros menores hasta tener problemas fáciles de resolver. A esta técnica de resolución de problemas se le denomina *divide y vencerás*.

Programación Modular

Para aplicar correctamente la programación modular es necesario tener en cuenta ciertos criterios:

- Cada módulo debe tener un único punto de entrada y un único punto de salida.
- Cada módulo debe realizar una única función bien definida.
- Debe comportarse como una caja negra, dependiendo únicamente de las entradas.
- No deben ser demasiado amplios (50 líneas de código como mucho).
- Máxima *cohesión*, mínimo *acoplamiento*.

Programación Modular

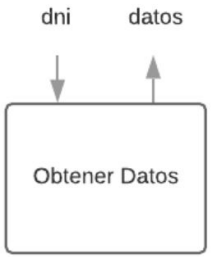
Cohesión

Cohesión de un módulo es el nivel de relación entre los elementos software (instrucciones, definiciones de datos o llamadas a otros módulos) que están contenidos en dicho módulo.

Existen diferentes tipos de cohesión, y no todos son deseables.

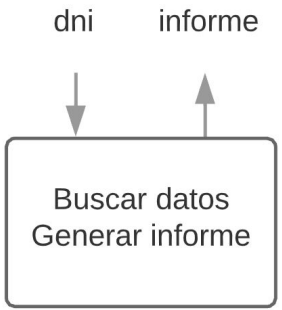
Programación Modular. Cohesión

- **Cohesión Funcional: Los elementos dentro del módulo contribuyen a la realización de una única función.**



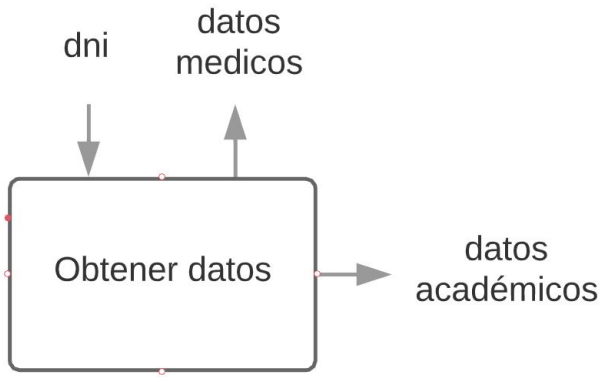
Programación Modular. Cohesión

- **Cohesión secuencial: Los elementos del módulo realizan varias tareas una detrás de otra de manera que la salida de una es necesaria para la entrada de la siguiente.**



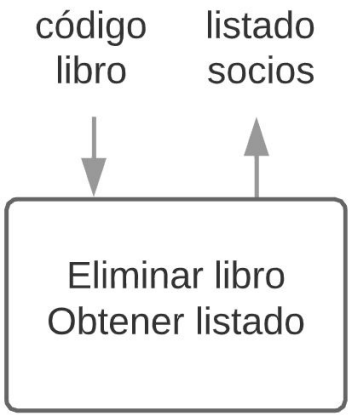
Programación Modular. Cohesión

- **Cohesión comunicacional: Contiene actividades paralelas que comparten los mismos datos.**



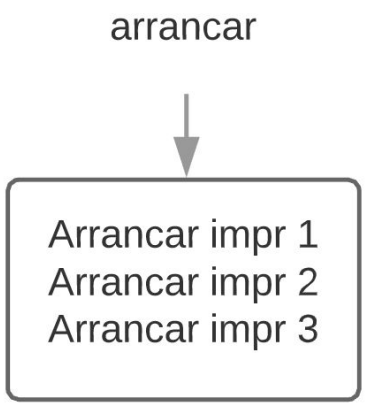
Programación Modular. Cohesión

- **Cohesión procedural: Contiene diferentes actividades no relacionadas entre sí. También es posible que los datos de entrada y salida no tengan relación entre sí.**



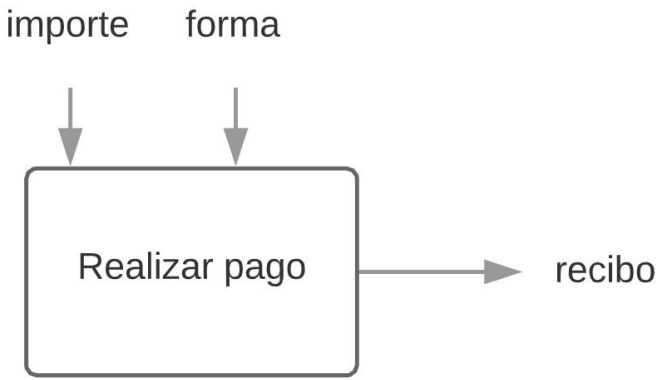
Programación Modular. Cohesión

- **Cohesión temporal: Las actividades se relacionan por el momento en el tiempo en el cual se llevan a cabo. Es posible que no utilicen ningún dato de entrada o salida.**



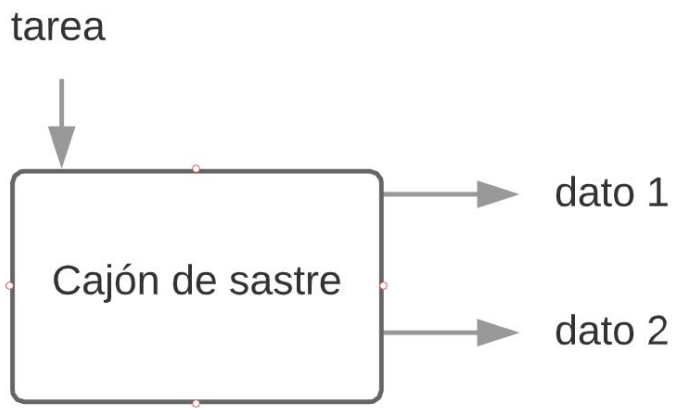
Programación Modular. Cohesión

- **Cohesión lógica: Se realizan actividades de una misma categoría general pero sin tener relación entre sí. Por ejemplo las librerías matemáticas.**



Programación Modular. Cohesión

- **Cohesión casual: No hay ninguna relación entre los elementos que lo componen sino que son fruto de una relación caótica.**



Mediante el flag tarea se indica qué se debe realizar. Algunas tareas devuelven datos y otras no

Programación Modular. Cohesión

No todos los tipos de cohesión son deseables. Es necesario mantener una cohesión fuerte para favorecer la reutilización y mantenimiento de los programas.

Cohesión fuerte:

- Funcional
- Secuencial
- Comunicacional

Cohesión débil:

- Procedural
- Temporal
- Lógica
- Casual

Programación Modular

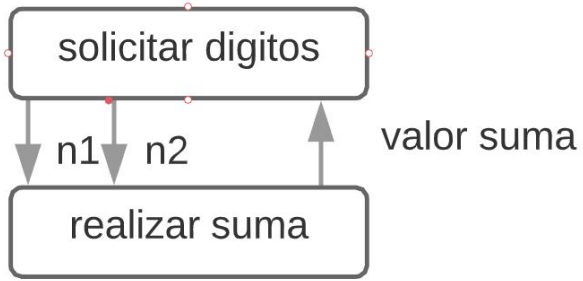
Acoplamiento

El término acoplamiento hace referencia al grado y forma de dependencia entre módulos. A menor cantidad de información compartida entre módulos menor será su acoplamiento y, por lo tanto, mejor será el diseño.

Existen diferentes tipos de acoplamiento.

Programación Modular. Acoplamiento

- **Acoplamiento normal: Se dan cuando un módulo a invoca a otro módulo b.**



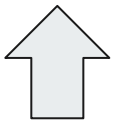
Programación Modular. Acoplamiento

- Acoplamiento externo: Se dan cuando dos o más módulos utilizan las mismas fuentes externas de datos (una interfaz de un programa externo, por ejemplo un puerto COM de una báscula).
- Acoplamiento global: Los módulos utilizan los mismos datos globales (una variable global, un fichero, una base de datos).
- Acoplamiento patológico: Se produce cuando un módulo lee o modifica los datos internos de otro módulo.

Programación Modular. Acoplamiento

En general hay que buscar un bajo acoplamiento en los programas, lo que favorece el mantenimiento del código ya que cada módulo debe ser una caja estanca independiente del resto y su modificación no debe afectar al resto del programa.

Acoplamiento Débil
(favorece el mantenimiento)



Acoplamiento Fuerte
(dificulta el mantenimiento)

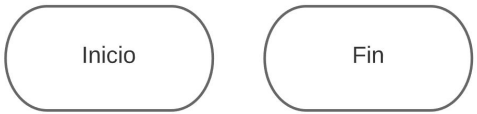
- Acoplamiento normal
- Acoplamiento externo
- Acoplamiento global
- Acoplamiento patológico

Diagramas de Flujo

Un diagrama de flujo, ordinograma o flujograma es una representación gráfica de un algoritmo o proceso.

Facilita la compresión del algoritmo gracias a a descripción visual que aporta sobre el flujo de ejecución.

Todo diagrama comienza y termina con un terminal, representado mediante un óvalo o elipse.

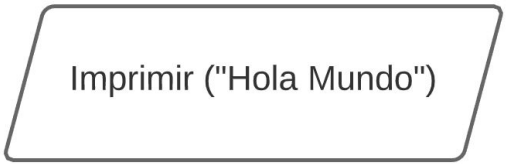


Diagramas de Flujo

La entrada de datos desde teclado se representa mediante un trapecio rectángulo. Indica la detención del proceso hasta que el usuario teclee los datos.

Diagramas de Flujo

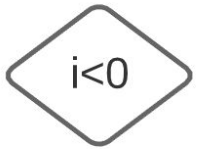
La comunicación con los periféricos de entrada y salida se representa mediante un paralelogramo.



El orden de las operaciones se realiza mediante flechas

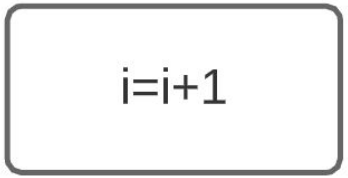
Diagramas de Flujo

Las decisiones se representan con un rombo, del que salen tantas líneas de flujo como alternativas posibles



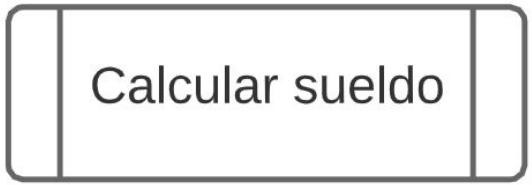
Diagramas de Flujo

Los procesos que llevan a cabo las operaciones internas de cálculo se representan mediante un rectángulo



Diagramas de Flujo

Si la tarea del diagrama es compleja y no puede ser representada en un único proceso se podrá dividir el diagrama en subprocesos, definidos en otro lugar y representados por un rectángulo con doble línea en cada lado.



Diagramas de Flujo

Las bases de datos con las que se intercambia información tienen su propio símbolo

Al igual que los ficheros

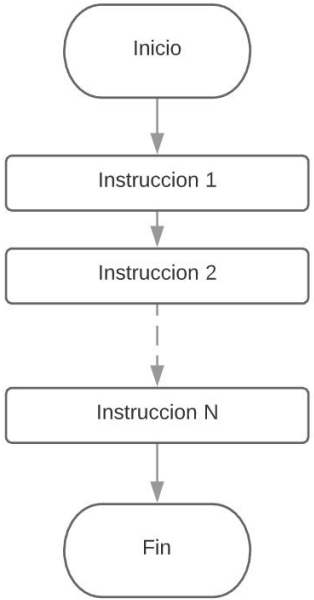
Diagramas de Flujo

Operadores

Aritméticos		Relacionales		Lógicos	
+	Suma	=	igual	and	y lógico
-	Resta	<	menor	or	o lógico
*	Multiplicación	<=	menor o igual	not	negación lógica
/	División real	>=	mayor o igual	Especiales	
div	División entera	<>	distinto	← o =	Asignación
mod o %	Resto				
^	Potencia				

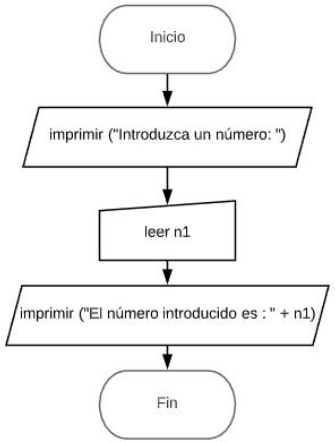
Diagramas de Flujo. Estructuras de control

Secuencial



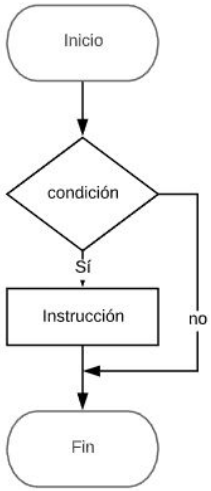
Diagramas de Flujo. Estructuras de control

Ejemplo: Leer un número introducido por pantalla y mostrarlo al usuario

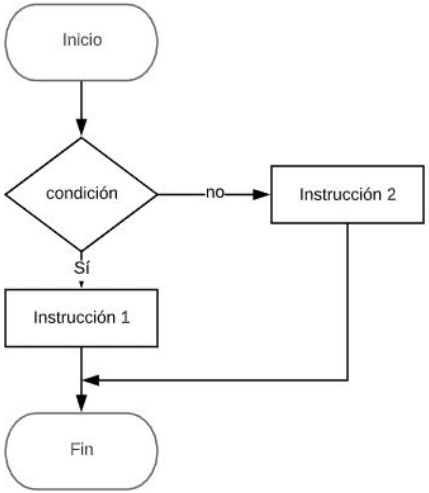


Diagramas de Flujo. Estructuras de control

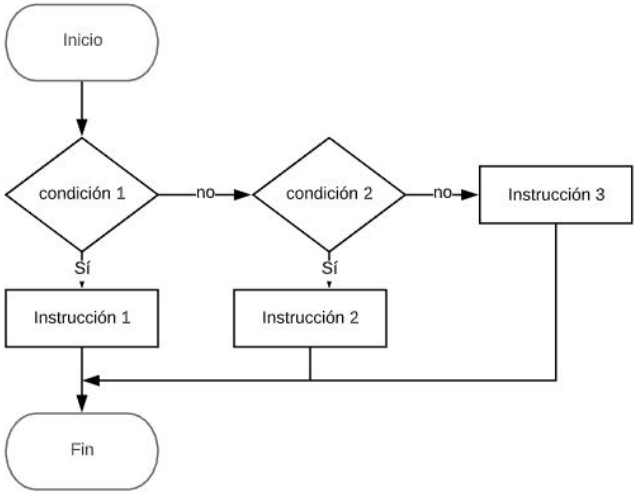
Alternativa simple



Alternativa doble

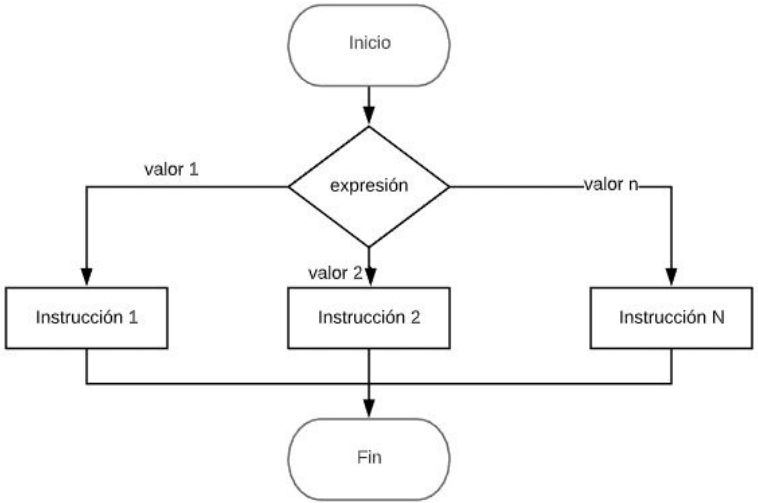


Alternativas simples anidadas



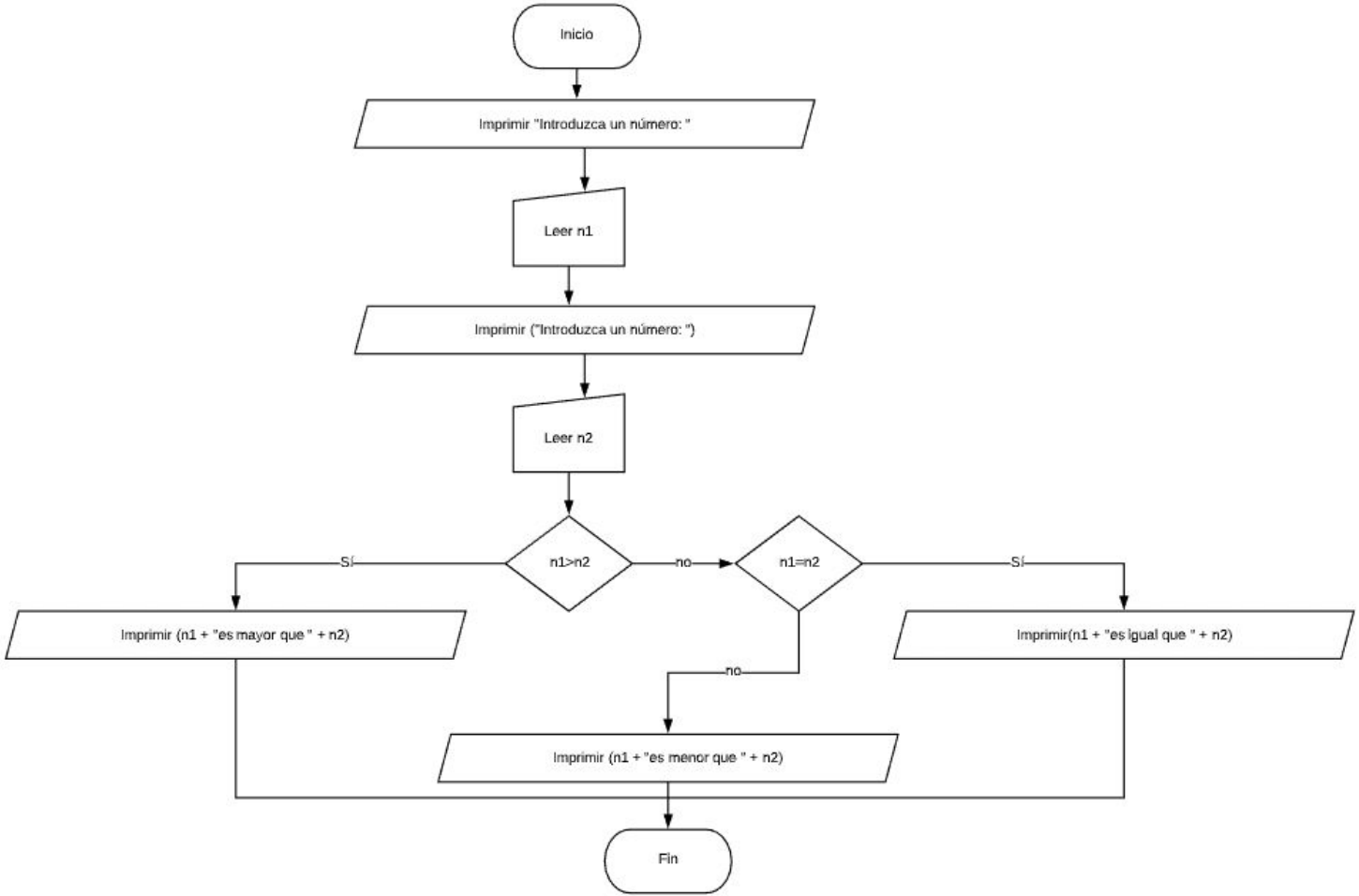
Diagramas de Flujo. Estructuras de control

Alternativa múltiple



Diagramas de Flujo. Estructuras de control

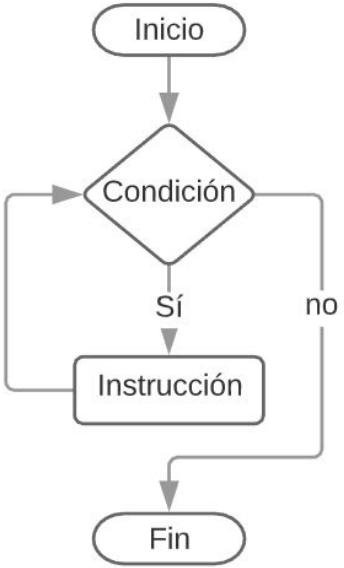
Ejemplo: Escribir el diagrama de flujo de un programa que, pidiendo dos números, indique cual es mayor, cual es menor o si son iguales


Inicio
Imprimir "Introduzca un número: "
Leer n1
Imprimir ("Introduzca un número: ")
Leer n2
n1>n2
Sí
no
n1=n2
Sí
Imprimir (n1 + "es mayor que " + n2)
Imprimir(n1 + "es igual que " + n2)
no
Imprimir (n1 + "es menor que " + n2)
Fin

Diagramas de Flujo. Estructuras de control

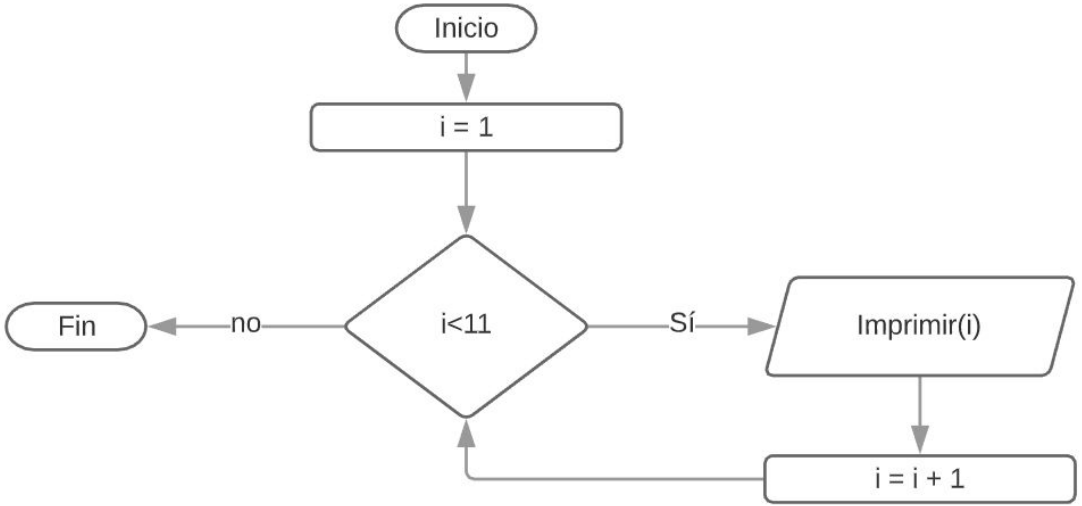
Iterativa While

El código se repite mientras se cumpla una condición dada pero sólo si se ha cumplido al menos una vez (primero se evalúa la condición)



Diagramas de Flujo. Estructuras de control

Ejemplo: Escribir, mediante un diagrama de flujo, la estructura de un programa que imprima los números del 1 al 10 utilizando un bucle While.


Inicio
i = 1
i<11
no
Fin
Sí
Imprimir(i)
i = i + 1

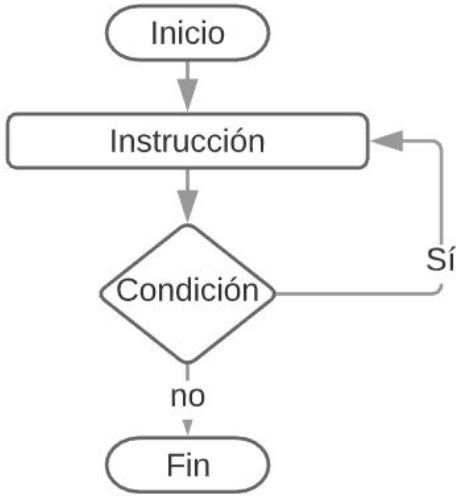
Diagramas de Flujo. Estructuras de control

Iterativa Do While

El código se repite mientras se cumpla una condición dada

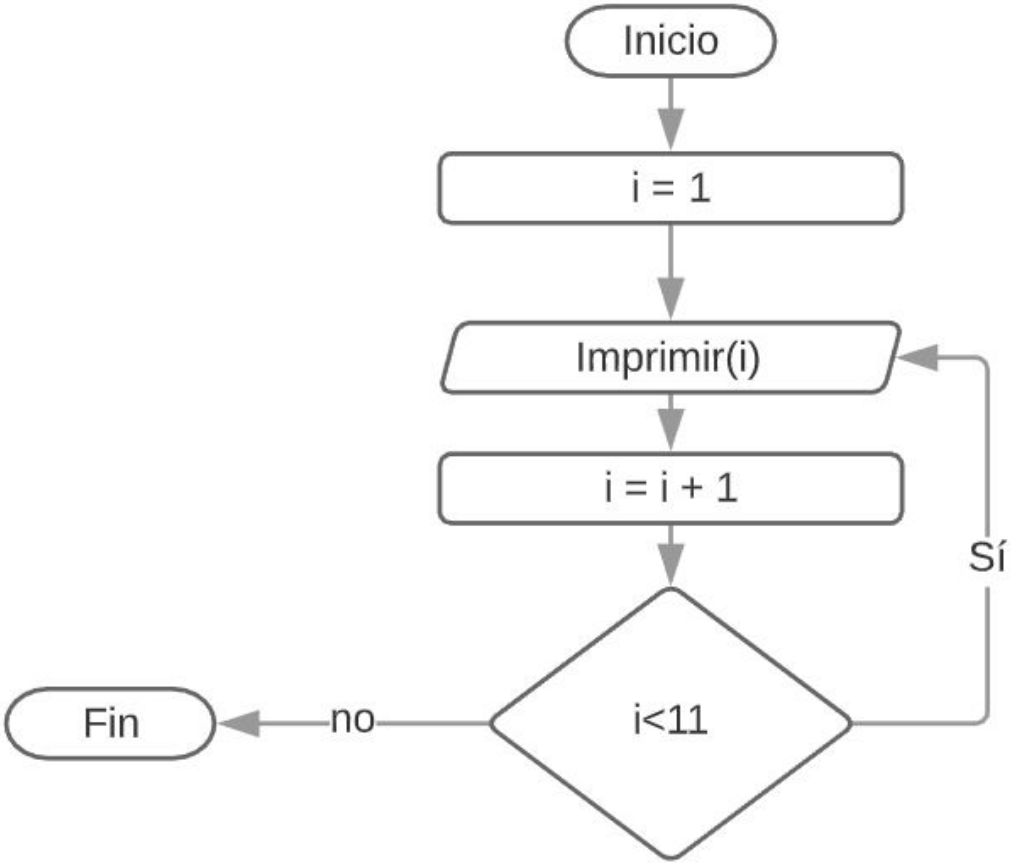
La primera vez siempre se ejecuta, ya que la condición se evalúa con posterioridad

Existe una iterativa similar (Repeat Until) que se repite hasta que se cumple la condición (los valores si/no de la condición están al revés que en el Do While)



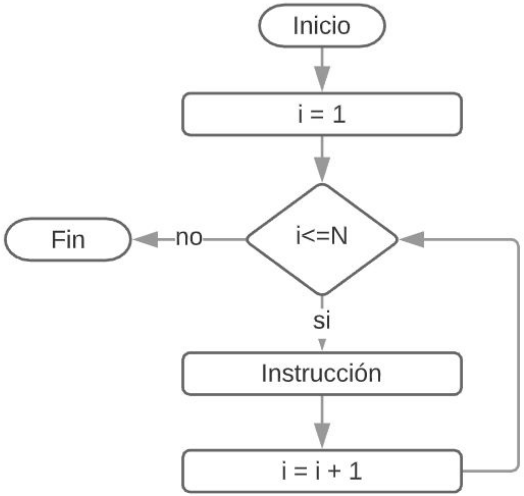
Diagramas de Flujo. Estructuras de control

Ejemplo: Escribir, mediante un diagrama de flujo, la estructura de un programa que imprima los números del 1 al 10 utilizando un bucle Do While.

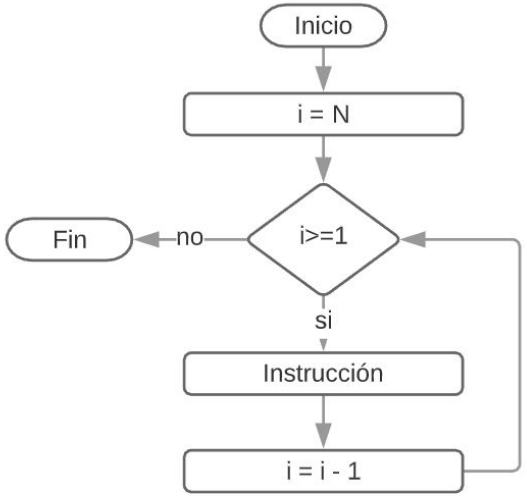

Inicio
i = 1
Imprimir(i)
i = i + 1
i<11
Sí
no
Fin

Diagramas de Flujo. Estructuras de control

Bucle For



Incremental



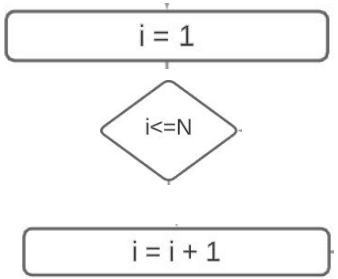
Decremental

Diagramas de Flujo. Estructuras de control

Bucle For

Analizando los bucles for, éstos se componen de tres elementos:

- Inicialización: Valor inicial que se le da a la variable de control
- Condición que se debe cumplir para mantenerse en el bucle
- Acción de incremento o decremento sobre la variable de control para terminar saliendo del bucle



Diagramas de Flujo. Procedimientos y Funciones

Los procedimientos y las funciones son estructuras modulares que se utilizan para separar las tareas a realizar dentro de un programa y así facilitar la lectura del diagrama.

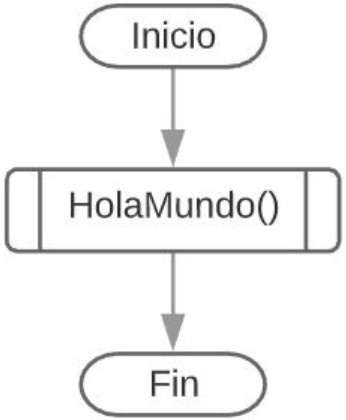
La diferencia entre ambas es que el procedimiento no devuelve ningún dato al flujo principal del programa mientras que las funciones siempre devuelven algún dato necesario en el flujo principal.

Tanto los procedimientos como las funciones pueden aceptar datos pasados como parámetro que se envían desde el flujo principal y se utilizan dentro del procedimiento o función.

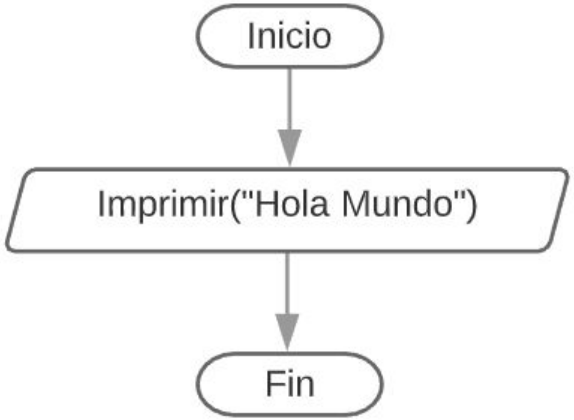
Diagramas de Flujo. Procedimientos

En el caso de usar un procedimiento deberá llamarse mediante el símbolo 

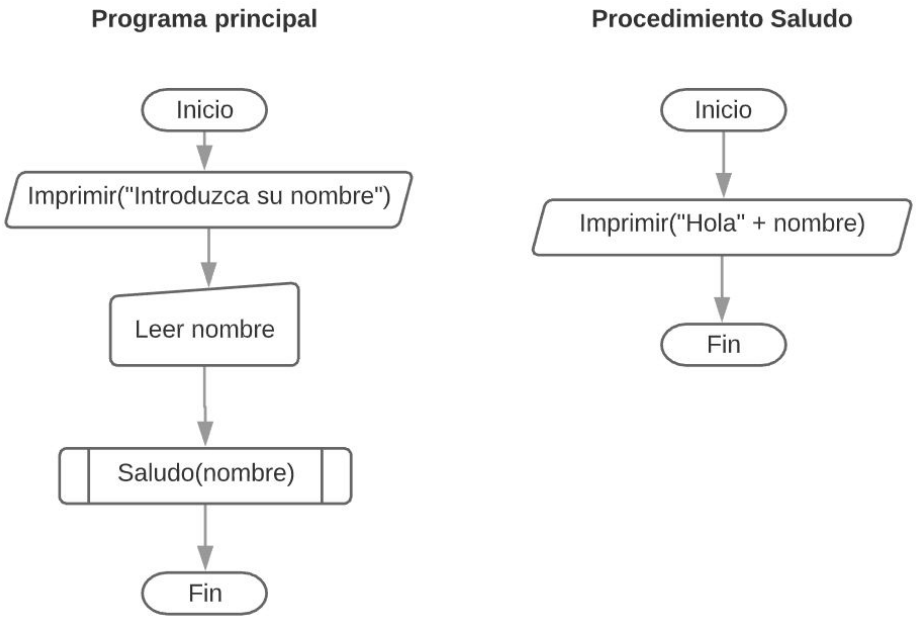
Programa principal



Procedimiento HolaMundo

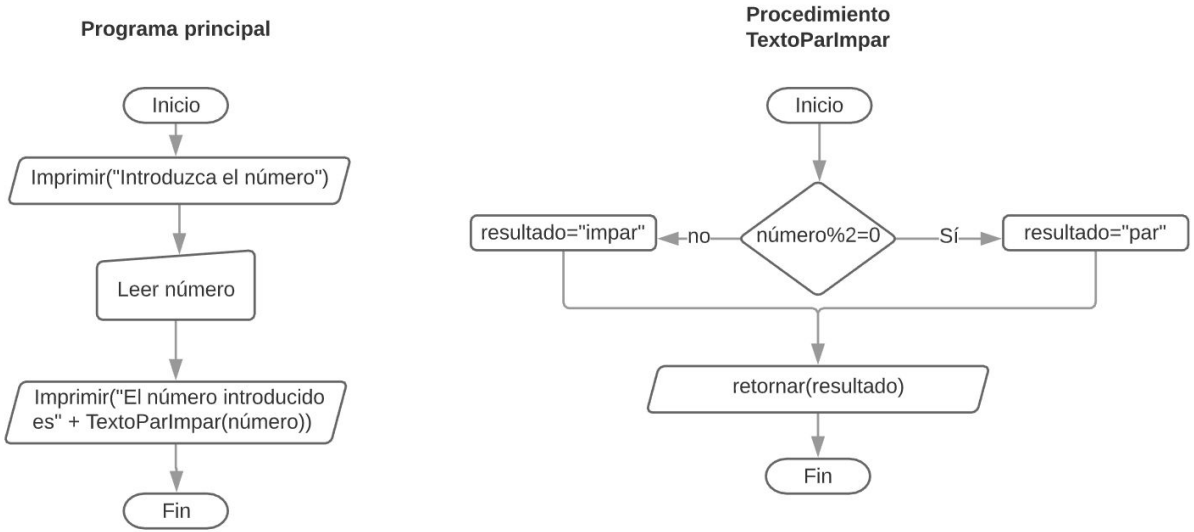


Diagramas de Flujo. Procedimientos

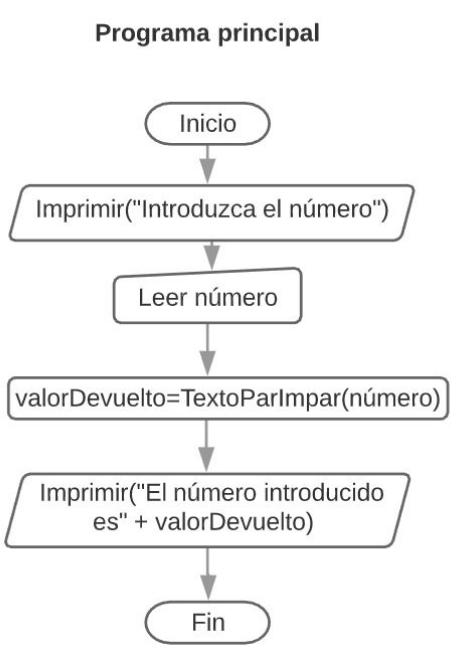


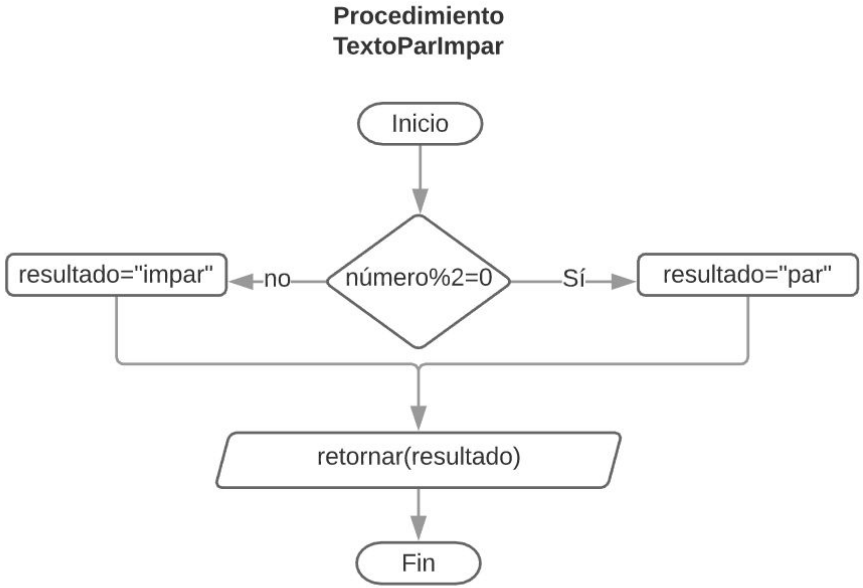
Diagramas de Flujo. Procedimientos

En el caso de usar una función basta con indicar la llamada en el proceso



Diagramas de Flujo. Procedimientos





Se puede realizar también una asignación intermedia del valor devuelto a una variable llamada valorDevuelto, que después imprimimos