Cuando empezamos a programar necesitamos hacer dos cosas, la primera adaptar nuestro tren de pensamiento y así poder describir los pasos necesarios para resolver cierto problema. Y la segunda escribir esos pasos en un lenguaje de programación que pueda entender un computador. En ocasiones puede ser una tarea un poco desesperante hacer las dos cosas al mismo tiempo, sobre todo al comenzar. Para facilitar un poco nuestras vidas podemos echar mano del pseudocódigo algo que se ve y se comporta (un poco) como código, pero no lo es.

¿Por qué digo que parece código, pero no es? Porque recuerda la sintaxis de algunos lenguajes, pero no es tan estricto como estos, está hecho para que nosotros los humanos podamos expresar ideas de forma un poco más parecida al lenguaje de programación.

Un algoritmo expresado en pseudocódigo se ve más o menos así:

```
Inicio
   a = 0;
   b = 0;
   imprimir "Introduzca el primer número"
   leer -> a
   imprimir "Introduzca el segundo número"
   leer -> b
   c = a + b
   imprimir "La suma de " a " y " b es " c
Fin
```

Hablemos un poco del algoritmo:

- Es relativamente seguro decir que el algoritmo describe el proceso de sumar dos números.
- Al principio del algoritmo asignamos el valor a las dos variables que se usarán en el, a esto se le llama *inicializar variables* y lo hacemos para que dichas variables comiencen con valores controlados por nosotros.
- Podemos notar que el algoritmo está delimitado por las palabras **Inicio** y **Fin**.
- Podemos notar que luego del inicio del algoritmo **indentamos** las siguientes instrucciones. Esto, para recalcar el hecho de que formar parte de un **bloque**.
- Con la palabra **imprimir** denotamos una salida, en este caso los mensajes que nuestro algoritmo muestra al usuario son: *"Introduzca el primer número"*, *"Introduzca el segundo número"* y *"La suma de " a " y " b " es " c*, luego comentaremos algo más sobre este último.
- La palabra **leer** la usamos para denotar que nuestro algoritmo "lee" la entrada que le suministra un usuario y la almacena en la memoria. La forma como denotamos que lo escrito por el usuario es guardado en la *variable* es mediante la flecha ->, como indicando que los datos entran en la *variable*. Luego hablaremos un poco más sobre la asignación a memoria.
- Cuando programamos cambiamos un poco la forma en que hacemos las fórmulas matemáticas, en este caso estamos diciendo que el resultado de la operación matemática *a + b* será guardado en la variable *c.*
- Finalmente, le mostramos al usuario el resultado de la operación, en este caso tenemos una mezcla de texto fijo y variables, lo encerrado entre comillas se considera texto fijo y lo que no son las variables. Cuando se muestre el mensaje de salida al usuario las etiquetas de las variables serán sustituidas por sus valores, por ejemplo, si el usuario introdujo un valor de 4 para a y un

valor de 1 para b el mensaje quedaría de la siguiente forma: *La suma de 4 y 1 es 5*.

Esto es todo, nuestro algoritmo espera que el usuario le proporcione dos números, los sumará y mostrará al usuario de vuelta el resultado.

Asignaciones a memoria

Como nosotros los humanos no manejamos la información de la misma forma de los computadores, preferimos asignar etiquetas a la parte de la memoria donde queremos guardar información y a estas etiquetas le llamamos variables. Así podemos referirnos a los datos guardados en la misma de forma relativamente fácil.

Las variables pueden tener cualquier etiqueta siempre y cuando comience con una letra.

Cuando queremos que el usuario nos proporcione datos usamos la instrucción **leer**:

leer -> variable

El algoritmo se “detiene” y espera por la entrada del usuario. Y dicha entrada sera almacenada en la variable.
Es una buena práctica de programación asignar valores por defecto a las variables que se van a usar al principio del programa o algoritmo. Esto nos asegura que no toman valores aleatorios que ya estuviesen en la memoria del computador.

Estructuras condicionales

Nuestro algoritmo fue bastante lineal, hizo una sola cosa, tuvo un comienzo y un solo final. Pero podemos complicarlo un poco más y ser más expresivos y especificar mejor nuestro algoritmo. ¿Que pasa si el usuario introduce algo que no es un número? Veamos esta nueva versión de nuestro algoritmo:

```
Inicio
  a = 0;
  b = 0;
  imprimir "Introduzca el primer número"
  leer -> a
  si a no es un numero entonces
    imprimir a " no es un valor válido, no puedo continuar."
    ir a fin
  fin si
  imprimir "Introduzca el segundo número"
  leer -> b
  si b no es un numero entonces
    imprimir b " no es un valor válido, no puedo continuar."
    ir a fin
  fin si
  c = a + b
  imprimir "La suma de " a " y " b es " c
Fin
```

Listo, ahora nuestro algoritmo se enfrentará a decisiones, se cuestionará que tipo de información le ha suministrado el usuario, y tomara un curso de acción dependiendo del resultado lógico del anterior cuestionamiento.

Dicho coloquialmente, verificara que la entrada del usuario es la que se quiere y si no lo es mostrara un mensaje de error y saldrá.

Terminar el algoritmo de esta manera es un tanto extremo, pero luego veremos una forma de hacer el algoritmo más amigable.

Veamos ahora la anatomía de una estructura de decisión:

si premisa_logica **entonces**
  Bloque de código que se ejecuta si la premisa
  lógica es cierta
fin si

Traducido quiere decir, si la premisa lógica resulta ser cierta realiza lo siguiente. Es como si por ejemplo, la mamá le dice a uno:

— Mijo, vaya a la bodega y si el queso está a menos de 100 bolívares, compre un kilo.

si precio_queso < 100 **entonces**
  Comprar queso
fin si

Como puede verse, el precio del queso en cuestión, es menor a 100 o no lo es. La evaluación lógica da como resultado *verdadero* o *falso*. Si la evaluación resulta verdadera se procede a comprar el queso, de lo contrario se regresa uno de la bodega con las manos vacías.

En otras ocasiones es necesario hacer algo distinto cuando la premisa lógica no es verdadera.

si precio_queso < 100 **entonces**
  Comprar 1Kg de queso
de lo contrario
  Comprar 1/4Kg de queso
fin si

En este ejemplo, vamos a la bodega y si el precio del queso es menor a 100 compramos un kilo, en cualquier otro caso compramos 1/4 de kilo.

La forma general de escribir esta estructura de decisión es la siguiente:

si premisa_logica **entonces**
  Bloque de código que se ejecuta si la premisa
  lógica es cierta
de lo contrario
  Bloque de código que se ejecuta si la premisa
  lógica es falsa.
fin si

Es posible evaluar más condiciones con el **si**, con el siguiente ejemplo podemos verlo:
*Colocar un **de lo contrario** al final es completamente opcional.*

si precio_queso < 100 **entonces**
  Comprar 2Kg de queso
de lo contrario si precio_queso == 100 **entonces**

Comprar 1Kg de queso

de lo contrario

Comprar 1/4Kg de queso

fin si

Nótese que la comprobación de igualdad lleva dos signos igual ==.

En este ejemplo tenemos tres opciones, si el precio es menor a 100 compramos 2 kilos, si es igual a 100 compramos 1 kilo y si ninguno de los dos anteriores es verdadero compramos un cuarto de kilo.

*Podemos tener tantos **de lo contrario si** como opciones tengamos que manejar.*

Estructuras de repetición

En los algoritmos y programas es necesario la mayoría de las veces repetir una acción o acciones sobre un conjunto de datos, o repetir una acción dependiendo del resultado de otra, o repetir una acción infinitamente.

Imaginemos que tenemos que caminar 5 de pasos para ir del punto *a* al punto *b*. Si escribimos las instrucciones como un algoritmo tendremos algo así:

Inicio

caminar un paso hacia adelante

caminar un paso hacia adelante

caminar un paso hacia adelante

caminar un paso hacia adelante

caminar un paso hacia adelante

Fin

Como son 5 pasos parece fácil. ¿Y si fuesen 100, ó 1000?

Cuando queremos representar la repetición de tareas con un límite conocido usamos la estructura de repetición **para,** así el algoritmo anterior queda:

```
Inicio
  i = 0
  para i = 0 hasta i = 4 hacer
    caminar un paso hacia adelante
  fin para
Fin
```

El **para** es un tipo de estructura de repetición definida. La estructura **para**hace uso de una variable para llevar la cuenta del avance de pasos, llamamos a este tipo de variables *contador.*

Con el para podemos contar de forma incremental o de forma decremental

```
p = 0
para p = 20 hasta p = 0 hacer
  imprimir "Faltan t -" p " segundos para el despegue"
fin para
```

Podemos contar en incrementos distintos a 1 o en decrementos distintos a 1

```
p = 0
para p = 2 hasta p = 10 con p = p + 2 hacer
  imprimir "El número " p " es par"
fin para
```

¿Como podríamos escribir un ciclo definido para que cuente de forma decremental en pasos de 7?

Existen otro tipo de estructuras de repetición, que son *indefinidas*, es decir no se manejan con una variable contador que se evalúa para determinar cuando termina el ciclo. Estas estructuras dependen de la evaluación de una premisa lógica para continuar iterando.

Una de estas estructuras de repetición se denomina **repita mientras**. El **repita mientras** ejecuta un bloque de código hasta que la premisa lógica deja de ser verdadera. No existe una cantidad de repeticiones definidas, se basa unicamente en un estado lógico, si nunca se produce un cambio en el estado de la premisa el ciclo continuara indefinidamente.

Podemos ilustrar un caso donde el uso del ciclo repita mientras, imaginemos que necesitamos vaciar un gran recipiente con agua haciendo uso uno mas pequeño, el pseudcódigo quedaría de la siguiente forma:

repita mientras recipiente_grande no vacio
  introducir recipiente_pequeño en recipiente_grande
  llenar recipiente_pequeño
  sacar recipiente_pequeño
  vaciar recipiente_pequeño
fin mientras

En este caso la premisa lógica es *recipiente_grande no vacio,* si recipiente grande no esta vacío la evaluación dará verdadero, en el momento que se vacíe la evaluación dará falso y el bucle o ciclo se detendrá.

Nótese que para que el ciclo comience el valor de la premisa lógica debe ser verdadero, de lo contrario el ciclo ni siquiera arrancara. Esto quiere decir que un ciclo **repita mientra**s puede ejecutarse cero o mas veces, en el ejemplo anterior si el recipiente grande esta vacío el ciclo no se ejecutara ni una sola vez.

Nos queda una estructura de repetición indefinida mas y para explicarla regresaremos a nuestro ejemplo de la suma de dos números. Una falla de diseño básica es que el algoritmo termina cuando el usuario no introduce un valor dentro del rango de lo esperado, vamos a corregir esto. Lo ideal es que el algoritmo pregunte una y otra vez al usuario hasta que este le suministre valores dentro de los parámetros del algoritmo. Para esto vamos a usar un tipo de estructura de repetición que se llama **repita hasta**:

```
Inicio
  a = 0;
  b = 0;
  repita
    imprimir "Introduzca el primer número"
    leer -> a
    si a no es un numero entonces
      imprimir a " no es un valor válido, intente de nuevo."
    fin si
  hasta a es un numero
  repita
    imprimir "Introduzca el segundo número"
    leer -> b
    si b no es un numero entonces
      imprimir b " no es un valor válido, intente de nuevo."
    fin si
  hasta b es un numero
  c = a + b
```

```
   imprimir "La suma de " a " y " b " es " c
Fin
```

El ciclo **repita hasta** se ejecuta, hasta que la condición lógica evaluada al final, en este caso, *a es un numero*. Sea cierta. Esto permite repetir una y otra vez la pregunta si el usuario insiste en suministrar datos erróneos.

Obviamente eliminamos la linea que dice **ir a fin**, puesto que no queremos terminar el programa, queremos que si el usuario introduce algo que no es un número el algoritmo le diga que el valor no es válido y le permita al usuario intentarlo de nuevo.

La estructura básica de un **repita hasta** es la siguiente:

```
repita
   Bloque de código que queremos repetir.
hasta premisa_logica
```

Como podemos ver el ciclo repita hasta se ejecuta al menos una vez, debido a que la premisa lógica se evalúa al final.

Glosario

- **Fin:** palabra de pseudocódigo para indicar el final del algoritmo o de un bloque.
- **imprimir:** palabra de pseudocódigo para indicar que el algoritmo va mostrar información al usuario.
- **indentación:** significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y mejor distinguirlo del texto adyacente.

- **Inicio:** palabra de pseudocódigo para indicar el inicio del algoritmo o de un bloque.
- **leer:** palabra de pseudocódigo para indicar que el algoritmo tomará información suministrada por el usuario, usualmente dicha información se guardará en memoria.
- **si:** estructura de decisión, que permite que el algoritmo actúe de forma distinta basado en el resultado de una evaluación lógica.
- **para**: estructura de repetición *definida* que repite un bloque de código o instrucciones una cantidad predeterminada de veces. Se llama definida porque la cantidad de veces que se ejecutara el bloque se define al momento de comenzar a ejecutarla. Esta estructura de repetición cuenta con una variable contador que a la que se le evalúa el valor para determinar si se termina el ciclo o no. También posee una manera de incrementar o decrementar dicha variable de forma automática en deltas definidos.
- **repita mientras**: es una estructura de repetición *indefinida* que que ejecuta un bloque de código o instrucciones mientras una premisa lógica da verdadero. En esta estructura la premisa lógica se evalúa al principio. De resultar esta falsa el ciclo no se ejecutara, quiere decir que este ciclo se ejecuta cero o más veces.
- **repita hasta**: es una estructura de repetición *indefinida* que que ejecuta un bloque de código o instrucciones hasta que una premisa lógica da verdadero. En esta estructura la premisa lógica se evalúa al final. De resultar esta verdadera el ciclo se detendrá, quiere decir que este ciclo se ejecuta una o más veces.