Programación 3 - **Estructuras de Repetición**

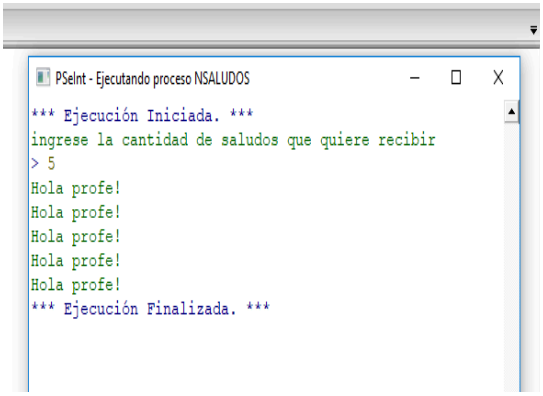
Que pasa cuando necesitamos hacer muchas veces algo? Siempre está la alternativa de escribir muchas veces la misma instrucción o primitiva pero no es óptimo ni fácil de documentar. Si tenemos que hacer un algoritmo para usar con 4 compañeros, podemos definir 4 variables, si tenemos que hacer un algoritmo para toda la división podremos crear 30 variables, pero para todo el colegio? Vamos a crear miles de variables? Una fiaca! La solución? Las **estructuras de repetición**.

Son varias las *estructuras de repetición* que existen en PSeInt, y están representadas por las primitivas:

- **Repetir**
- **Para**
- **Mientras**

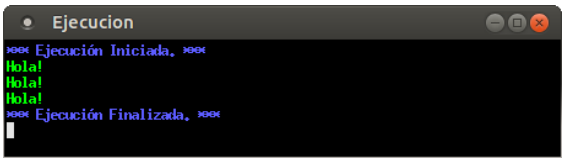
Veamos las tres alternativas de estructuras de repetición:

- Que PSeInt nos salude tantas veces como querramos

Seudocódigo	Salida
Algoritmo nSaludos Escribir "ingrese la cantidad de saludos que quiere recibir " Leer n cant<-0 Repetir Escribir 'Hola profe!' cant<-cant + 1 Hasta Que n == cant FinAlgoritmo	

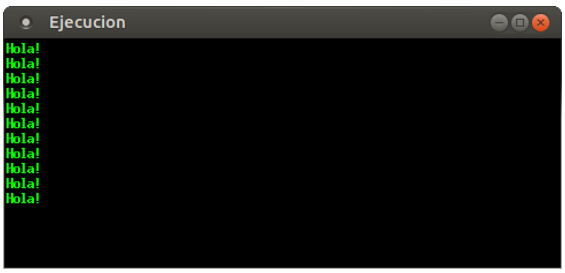
Lo que hace este algoritmo es saludar tantas veces como la cantidad de veces que el usuario haya ingresado que quiere recibir saludos. Cuando se cumple la condición, nos deja de saludar.

- Que PSeInt nos salude 3 veces:

Seudocódigo	Salida
Algoritmo TresSaludos **Para** indice **desde** 1 **hasta** 3 Escribir 'Hola!' **FinPara** FinAlgoritmo	

Lo que hace este algoritmo es saludarnos una cantidad fija de veces. Cuando llega a esa cantidad deja de saludarnos.

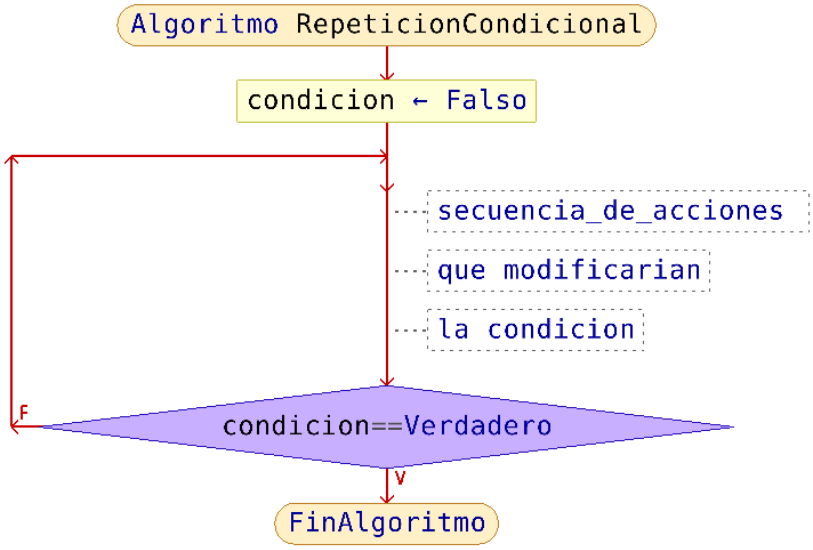
- Que PSeInt nos salude las veces que querramos, pero otra forma:

Seudocódigo	Salida
Algoritmo NSalu2 Escribir "Defini la cantidad de veces que queres que te salude" Leer cant nro<-1 Mientras nro <= cant Escribir 'Hola!' nro<-nro+1 FinMientras FinAlgoritmo	

Lo que hace este algoritmo es saludarnos mientras se cumpla la condición, cuando deja de cumplirse no nos saluda más.

Resumiendo las primitivas “Repetir”, “Para”, “Mientras”:

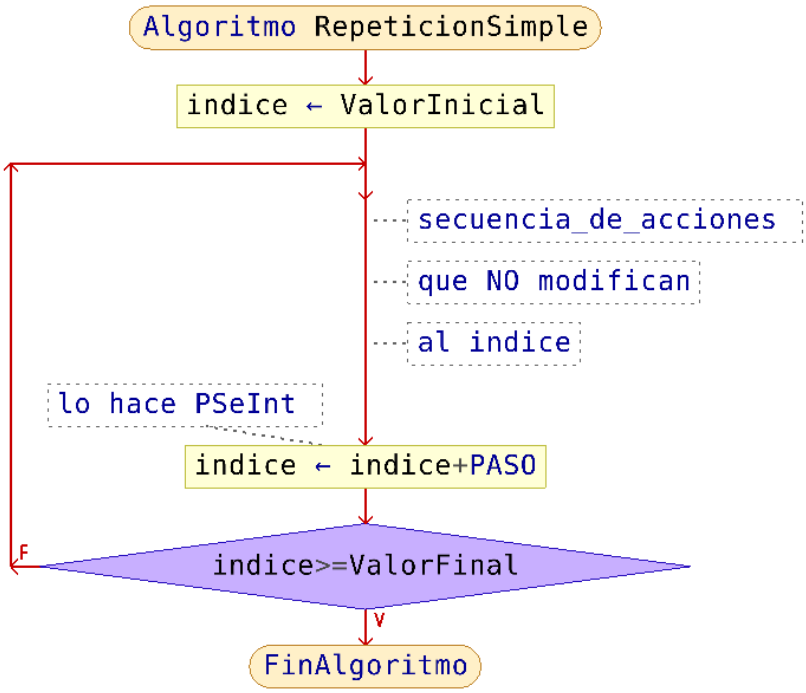
1) Estructura **Repetir:** PSeInt ejecuta secuencialmente lo que está programado entre las primitivas ***Repetir*** y ***Hasta Que*** (instrucciones indentadas). Luego evalúa la condición final, si ésta es VERDADERO, sale del ciclo. Pero si la condición final es FALSO vuelve a **iterar el bloque** (ejecutar la parte secuencial) y a evaluar. El ciclo *itera* indefinidamente hasta que se cumple la condición y ahí sale.



Primero hace un monton de cosas y al final evalua la condición.

2) Estructura **Para:** se parece a Repetir, pero con una condición de salida de *una determinada cantidad de veces*. Para ello, PSeInt lleva la cuenta con una **variable índice**, que sólo existe dentro del ciclo y es incrementada con cada **iteración** (cada ejecución del bloque). Ésta variable *índice* está sujeta a lo que el/la programadora decida:

    - valor final del *índice*: lo que tiene que valer para poder salir del bucle
    - valor inicial del *índice*: lo que valdrá en la primera *iteración*.
    - paso: valor que PSeInt suma al *índice*, antes de evaluar la condición de salida. Puede ser negativo, si por ejemplo queremos contar para atras.



3) La tercera primitiva es **Mientras**: se parece más todavía a *Repetir*, con dos diferencias:
   - la condición se evalúa al comienzo de la *iteración*
   - se rompe el ciclo con la condición en FALSO

   Primero evalua la condicion y si se cumple hace un monton de cosas.

   Mientras **y** ***Repetir*** se parecen al extremo. La diferencia radica en que si la condición es falsa en el caso del *mientras* la acción NUNCA se ejecuta la acción, mientras que en el *Repetir* se ejecutaría 1 vez. Esto se dá porque en el *Mientras* la condición se evalúa al inicio y en el *Repetir* se evalúa al final (o sea al menos una vez hace la acción que está programada).

Ejemplos de uso de las 3 estructuras de repetición:

- Cuantas veces nos saludará PSeInt, con éste algoritmo?

SEUDOCÓDIGO

```
Algoritmo EjHastaQue
     Repetir
          Escribir 'hola'
```

```
     Hasta Que 1==1
FinAlgoritmo
```

Tanto en un Mientras como en un Repetir, hay que garantizar que alguna de las instrucciones del cuerpo del ciclo cambie el valor de verdad de la condición. De lo contrario el bucle no finalizará, provocando lo que se conoce como **loop o ciclo infinito. Que se produzca esto esta mal, el programa nunca termina!!!**
Siempre el programador debe pensar estructuras iterativas que con alguna condicion salgan del ciclo repetitivo, o sea termine en algun momento.

VALIDACIONES

Validar un dato es controlar que el usuario no ingrese cualquier cosa, sino que sea un dato válido. Por ej si quiero saber el año de nacimiento, quiero que le dato sea numerico o entero. No quiero que ingrese dosmil tres en letras. Como puedo validar eso? Puedo usar definir, o bien con código si quiero validaciones lógicas, por ejemplo que el día de nacimiento no sea un negativo. Podemos programar esa lógica con distintas primitivas:

Por ej. usando la primitiva REPETIR:

SEUDOCÓDIGO

```
Algoritmo DiaDelMes
     Definir NUM como Numerico
     Repetir
        Escribir 'Ingrese día del mes (número):  '
        Leer NUM
     Hasta que (NUM > 0)  //no hay negativos en el calendario
     Escribir 'Ingresó el día: ',NUM
FinAlgoritmo
```

Como se te ocurre validar lo mismo pero usando la primitiva **MIENTRAS?**

La computadora ejecuta las instrucciones a una velocidad increíble y los resultados se ven instantáneamente, como si fuera una foto. Si queremos hacer un tiempo entre mensaje y mensaje podemos poner un **retardo**, como si estuviéramos jugando a las escondidas, es decir antes de volver a escribir el mensaje contar una cierta cantidad de veces, de la siguiente forma:

```
Algoritmo saludo3
        Escribir 'como te llamas?'
        Leer nombre
        Para x<-1 Hasta 5 Con Paso 1 Hacer
                Escribir x,' Hola, ' nombre
                Para t<-1 Hasta 10000000 Con Paso 1 Hacer
```

```
                    // no hacemos nada, solo contar
                Fin Para
        Fin Para
FinAlgoritmo
```

Observen que para cada Para hay un Fin Para.

Primitiva Mientras:

Que hace este código de abajo ?

SEUDOCÓDIGO

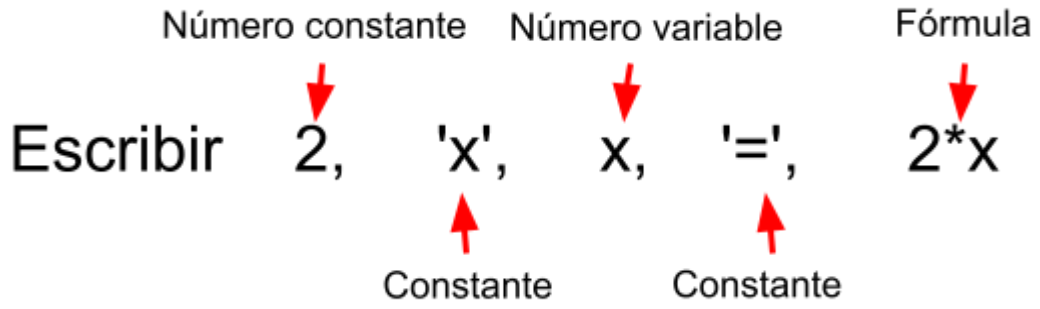
```
Algoritmo validacionEntrada
    Definir NUM como Numerico
    Escribir 'Ingrese un numero mayor a 0'
    Leer NUM
    Mientras no(NUM > 0) hacer
           Escribir 'Error. Ingrese un numero. Deber ser mayor a 0'
           Leer NUM
    FinMientras
    Mostrar 'Ingreso el numero: ',NUM
FinAlgoritmo
```

Poner no(Num >0) es lo mismo que poner Num <= 0, hay muchas formas de programar y obtener el mismo resultado. Recordemos que el NO niega lo que esta a continuacion.

Otro ejemplo de algoritmo usando la primitiva PARA:

```
Algoritmo tabla_del_dos
        Para x Desde 2 Hasta 20 Con Paso 2 Hacer
                Escribir x
        Fin Para
FinAlgoritmo
```

El algoritmo sólo nos muestra la sucesión de números pares (2 4 6 8 10 12 ...) NO la tabla del dos. Para realizar lo pedido debemos elaborar la salida de la siguiente forma:



En el Escribir las constantes y las variables se deben separar con comas (,). Los mensajes o frases son constantes que deben ser delimitados con comillas simples (‘ ‘). Los números y las fórmulas se escriben sin comillas, porque lo que se escribe es el valor que contiene esa posición de memoria.

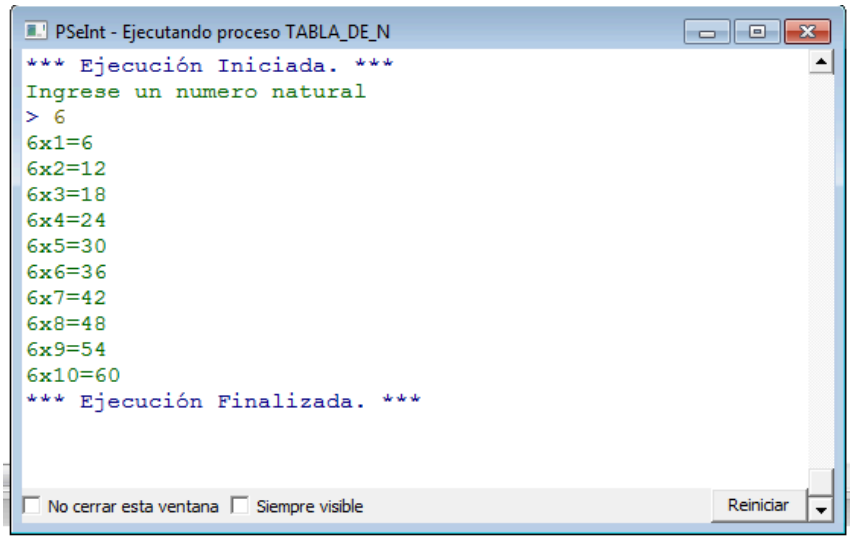
```
Algoritmo tabla_del_dos
        Para x<-1 Hasta 10 Con Paso 1 Hacer
                Escribir 2,'x',x,'=',2*x
        Fin Para
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso TABLA_DEL_DOS
*** Ejecución Iniciada. ***
2x1=2
2x2=4
2x3=6
2x4=8
2x5=10
2x6=12
2x7=14
2x8=16
2x9=18
2x10=20
*** Ejecución Finalizada. ***
```

Otro ejemplo más del PARA:

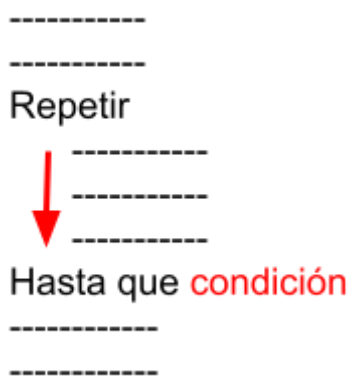
Generar la tabla de multiplicar de un número natural N, desde N = 1 hasta N = 10

```
Algoritmo Tabla_de_N
        Escribir 'Ingrese un numero natural'
        Leer n
        Para i<-1 Hasta 10 Con Paso 1 Hacer
                Escribir n,'x',i,'=',n*i
        Fin Para
FinAlgoritmo
```



Aparentemente el programa es correcto, pero ... si ingresamos valores negativos o decimales, que no responden con la consigna el programa corre igual. ¿Cómo podemos evitar esos números?

Debemos incorporar la estructura: Repetir_Hasta que, para que valide que el usuario ingrese solo números positivos. Sino va a seguir pidiendole el número.



Se ejecutarán repetidas veces las instrucciones que se encuentran dentro de la estructura mientras la condición sea Falsa, saliendo de la misma cuando sea Verdadera. Con estas instrucciones evitamos que sean números negativos. Tambien podemos validarlo con Mientras.

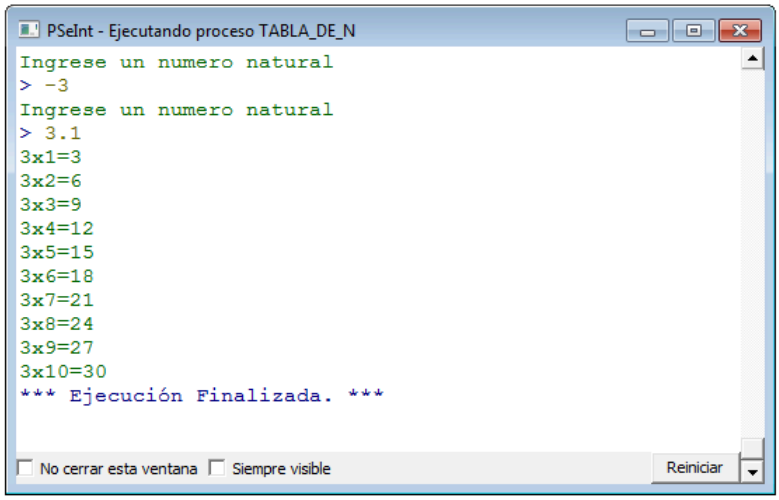
Para evitar que sean decimales, tenemos una opción que quita la porción decimal del número convirtiéndolo en un entero: la función Trunc. La función trunc tiene la siguiente forma:

TRUNC (X)

Es decir si X vale 2.25, luego de la instrucción se trunca a 2.

O sea utilizando ambas cosas estaremos evitando que ingresen a proceso valores negativos y decimales, y estaremos usando el entero del decimal ingresado.

```
Algoritmo Tabla_de_N
        Repetir
                Escribir 'Ingrese un numero natural'
                Leer n
                n<-trunc (n)
        hasta que n>0
        Para i<-1 Hasta 10 Con Paso 1 Hacer
                Escribir n,'x',i,'=',n*i
        Fin Para
FinAlgoritmo
```



Importante!
Validación de datos: es el proceso al que sometemos a los datos para verificar que sean correctos, de lo contrario no los procesamos.

Sigamos practicando:

Dado un número determinar si es par, o múltiplo de 2.
Existen dos formas que pueden determinar si un número es múltiplo de otro:

A - Si n/2 es igual a trunc(n/2) entonces n es múltiplo de 2.
B - Usando la función Mod, esta función devuelve el resto en una división entera, de la siguiente forma: n mod 2, devuelve el resto de la división entera n/2, pudiendo ser 0 (entonces n es múltiplo de 2) ó 1 (significa que n no es múltiplo de 2)

```
Algoritmo multiplo_de_2

        Escribir ' ingrese un nro'
        leer n
        si n/2= trunc(n/2) entonces
                escribir n,' es multiplo de 2'
        SiNo
                escribir n,' no es multiplo de 2'
        FinSi

FinAlgoritmo
```

Otra variante que tambien resuelve el problema:

```
Algoritmo multiplo_de_2

        escribir ' ingrese un nro'
        leer n
        si n mod 2= 0 entonces
                escribir n,' es multiplo de 2'
        SiNo
                escribir n,' no es multiplo de 2'
        FinSi

FinAlgoritmo
```