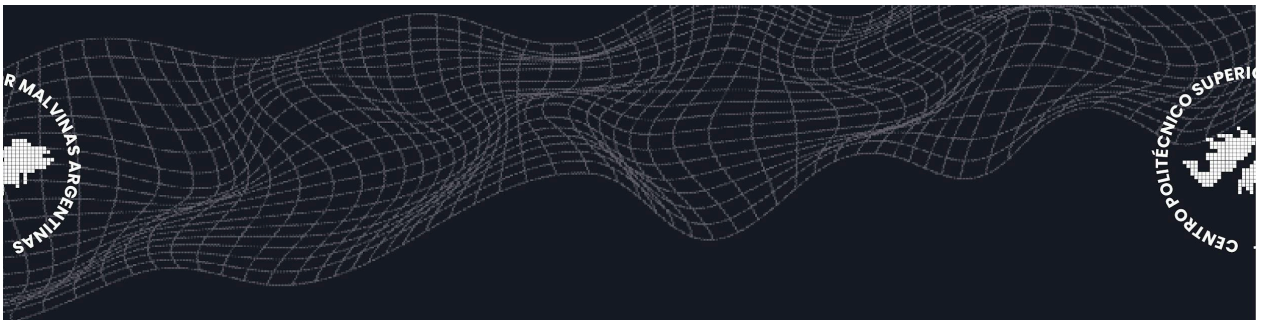


PROGRAMACIÓN 1

1° AÑO

CLASE Nº 2

ALGORITMOS, APLICACIONES,
ERRORES Y LENGUAJES DE PROGRAMACIÓN

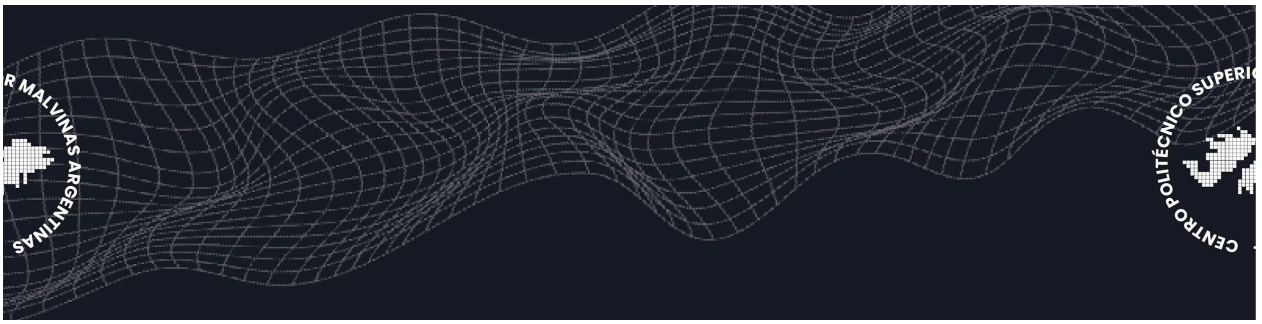
Contenido

En la clase de hoy trabajaremos los siguientes temas:

Algoritmos: Definición y características de los algoritmos, elementos que lo conforman y las fases en su creación. La notación para el diseño de algoritmos, diagramas de flujo, pseudocódigo. Estructuras para la creación de algoritmos, secuencial, de decisión y cíclica. Prueba de escritorio. Elementos de un programa, sentencias, comentarios, palabras reservadas, identificadores y líneas de preprocesador.

1. Presentación

¡Bienvenidos a la segunda clase de Programación 1! Donde veremos los algoritmos que son la base de la programación de computadoras. También, podremos conocer cómo están construidos internamente, mostrando las estructuras utilizadas para su construcción. Veremos cómo funcionan paso a paso entendiendo su comportamiento. ¡Iniciemos!



2. Desarrollo

ALGORITMOS

¿Qué es un algoritmo?

Según la RAE: *conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.*

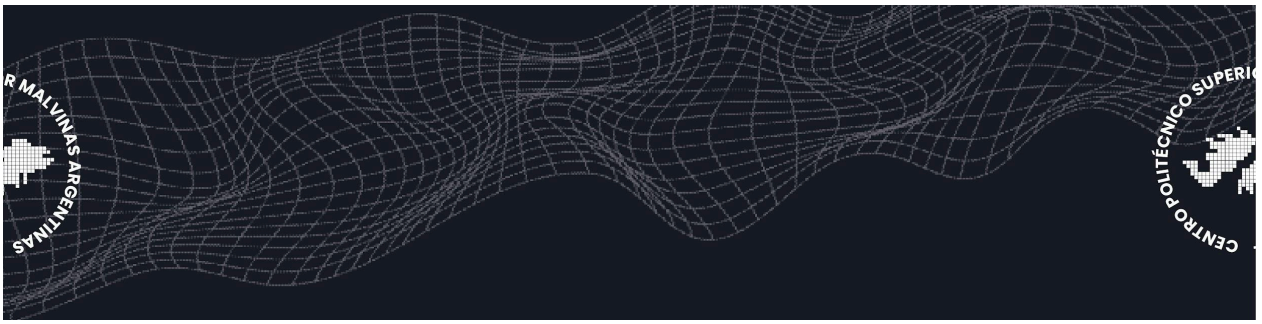
Los algoritmos, como indica su definición oficial, son una serie de pasos que permiten obtener la solución a un problema.

El lenguaje algorítmico es aquel que implementa una solución teórica a un problema indicando las operaciones a realizar y el orden en el que se deben efectuarse.

Para profundizar acerca de la creación de algoritmos te invitamos a ver el siguiente video:
https://www.youtube.com/watch?v=HcagN23sOH4&list=PLRb-cSJeUGHN08wBkBo1wpoeuk9IuHW-b&ab_channel=JuanRaGarciaMontes
(Programación: Conceptos básicos - JuanRa Garcia Montes)

Elementos que conforman un algoritmo

Entrada. Los datos iniciales que posee el algoritmo antes de ejecutarse.
Proceso. Acciones que lleva a cabo el algoritmo.
Salida. Datos que obtiene finalmente el algoritmo



Fases en la creación de algoritmos

Hay tres fases en la elaboración de un algoritmo:

Análisis. En esta se determina cuál es exactamente el problema a resolver. Qué datos forman la entrada del algoritmo y cuáles deberán obtenerse como salida.

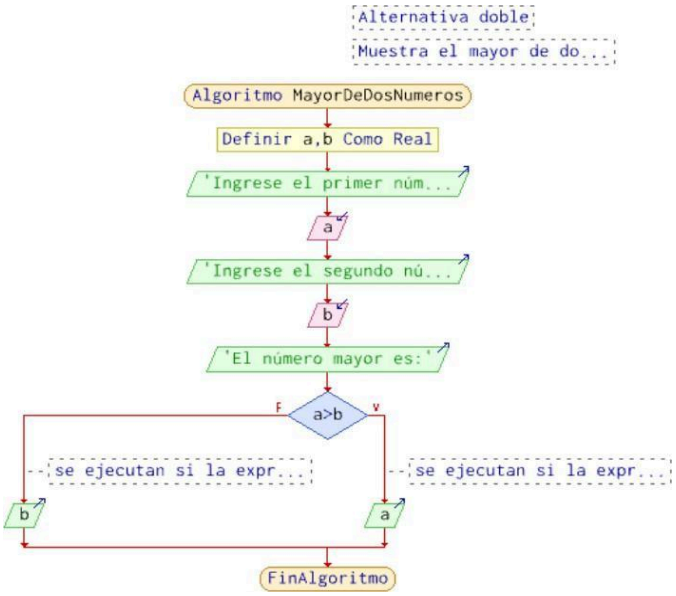
Diseño. Elaboración del algoritmo.

Prueba. Comprobación del resultado. Se observa si el algoritmo obtiene la salida esperada para todas las entradas.

Notaciones para el diseño de algoritmos

Diagramas de flujo

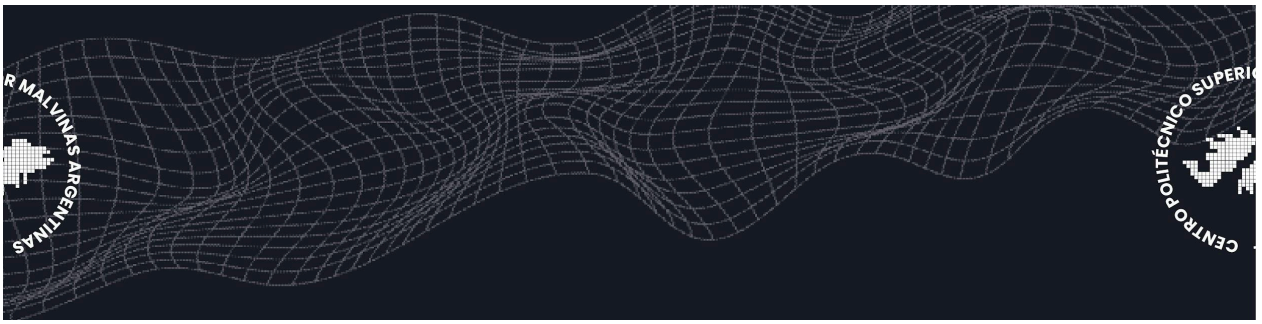
Un Ejemplo:



Las bases de la programación estructurada fueron enunciadas por Niklaus Wirdth. Según este científico cualquier problema algorítmico podía resolverse con el uso de estos tres tipos de instrucciones:

- ***Secuenciales***. Instrucciones que se ejecutan en orden normal. El flujo del programa ejecuta la instrucción y pasa a ejecutar la siguiente.
- ***Alternativas***. Instrucciones en las que se evalúa una condición y dependiendo si el resultado es verdadero o no, el flujo del programa se dirigirá a una instrucción o a otra.
- ***Iterativas.*** Instrucciones que se repiten continuamente hasta que se cumple una determinada condición.

```
// Alternativa doble
// Muestra el mayor de dos números leídos.
Algoritmo MayorDeDosNumeros
    Definir a,b Como Real;

    Escribir 'Ingrese el primer número:';
    Leer a;

    Escribir 'Ingrese el segundo número:';
    Leer b;

    Escribir 'El número mayor es:';

    Si a > b Entonces
        //se ejecutan si la expresión es verdadera
        Escribir a;
    SiNo
        //se ejecutan si la expresión es falsa
        Escribir b;
    FinSi

FinAlgoritmo
```



Comentarios

En pseudocódigo los comentarios que se deseen poner (y esto es una práctica muy aconsejable) se ponen con los símbolos // al principio de la línea de comentario (en algunas notaciones se escribe **). Cada línea de comentario debe comenzar con esos símbolos:

```
Algoritmo NombreDelAlgoritmo

      instrucciones;
      //comentario
      //comentario
      instrucciones;
FinAlgoritmo
```

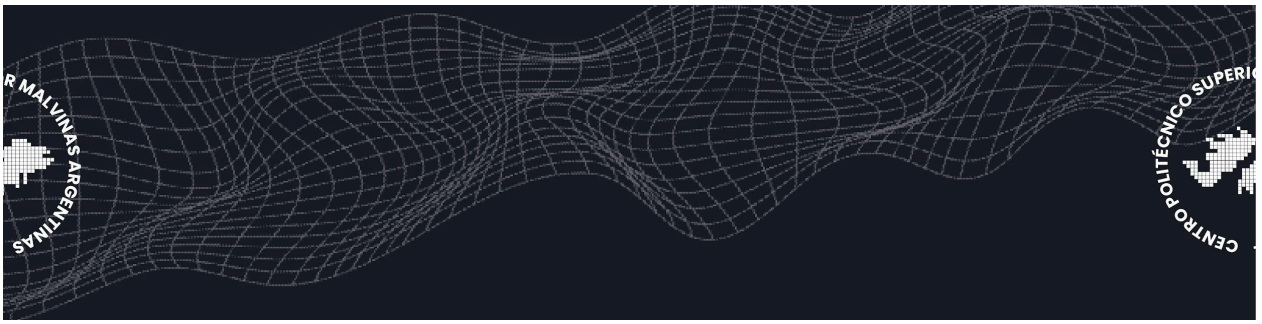
Creación de algoritmos

Instrucciones y tipos de instrucciones

El proceso de diseño del algoritmo o posteriormente la codificación consiste en definir las ACCIONES o INSTRUCCIONES que se deben ejecutar para resolver el problema.

Las instrucciones disponibles en un lenguaje de programación dependen de cada lenguaje específico. Las acciones o instrucciones se deben escribir en el mismo orden en que han de ejecutarse, es decir, en secuencia. La escritura de estas instrucciones sigue unas normas muy estrictas.

```
      Instrucción_1;
      Instrucción_2;
      ...
      Instrucción_n;
```



En este apunte veremos las instrucciones básicas que soportan todos los lenguajes. Que pueden ser de estos tipos:

- ***Primitivas***

Son acciones sobre los datos del programa. Asignación e Instrucciones de Entrada/Salida

- ***Declaraciones***

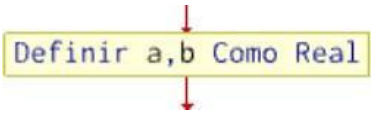
Obligatorias en el pseudocódigo, opcionales en otros esquemas. Sirven para advertir y documentar el uso de variables y subprogramas en el algoritmo.

- ***Control***

Sirven para alterar el orden de ejecución del algoritmo. En general el algoritmo se ejecuta secuencialmente. Gracias a estas instrucciones el flujo del algoritmo depende de ciertas condiciones que nosotros mismos indicamos.

Instrucciones de declaración

Se utilizan para indicar el nombre y las características de las variables que se utilizan en el algoritmo. Las variables son nombres a los que se les asigna un determinado valor y son la base de la programación. Al nombre de las variables se le llama identificador.

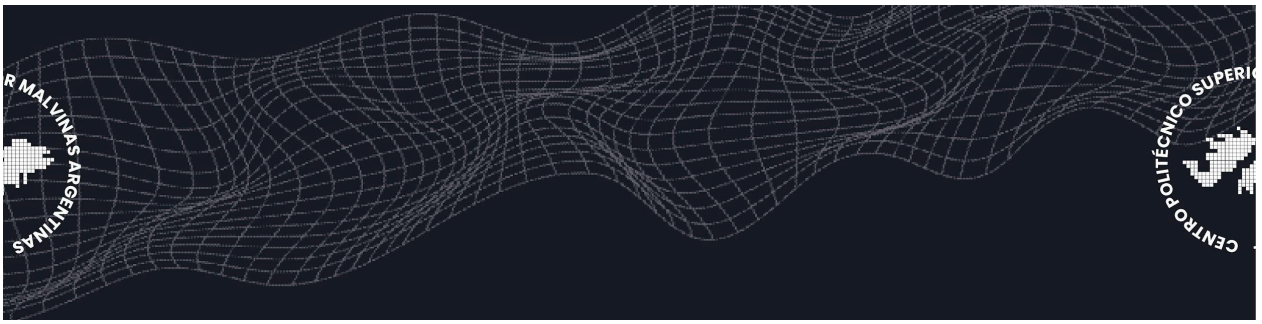


Acción de declaración en Diagrama de Flujo de Datos (DFD)

Identificadores

Los algoritmos necesitan utilizar datos. Los datos se reconocen con un determinado identificador (nombre que se le da al dato). Este nombre tiene las siguientes restricciones:

- Sólo puede contener letras, números y el carácter _
- Debe comenzar por una letra.



- No puede ser repetido en el mismo algoritmo. No puede haber dos elementos del algoritmo (dos datos por ejemplo) con el mismo identificador.
- Conviene que sea aclarativo, es decir que represente lo mejor posible los datos que contiene. x no es un nombre aclarativo, saldoMensual sí lo es.
- Se utiliza con frecuencia la notación Camel Case (https://es.wikipedia.org/wiki/Camel_case)

Declaración de variables

Es aconsejable al escribir pseudocódigo indicar las variables que se van a utilizar (e incluso con un comentario indicar para qué se van a usar). En el caso de los otros esquemas (diagramas de flujo y tablas de decisión) no se utilizan (lo que fomenta malos hábitos).

Esto se hace mediante la sección del pseudocódigo llamada Definir, en esta sección se colocan las variables que se van a utilizar. Esta sección se coloca antes del inicio del algoritmo. Y se utiliza de esta forma:

```
Algoritmo NombreDelAlgoritmo
Definir identificador1, identificador2 Como Real;
...
instrucciones;
FinAlgoritmo
```

El tipo de datos de la variable puede ser especificado de muchas formas, pero tiene que ser un tipo compatible con los que utilizan los lenguajes informáticos. Se suelen utilizar los siguientes tipos:

- Entero. Permite almacenar valores enteros (sin decimales).
- Real. Permite almacenar valores decimales.
- Carácter. Almacenan un carácter alfanumérico.
- Lógico (o booleano). Sólo permiten almacenar los valores verdadero o falso.
- Texto. A veces indicando su tamaño (texto(20) indicaría un texto de hasta 20 caracteres) permite almacenar texto. Normalmente en cualquier lenguaje de programación se considera un tipo compuesto.

También se pueden utilizar datos más complejos, pero su utilización queda fuera de este apunte de clase.

Constante

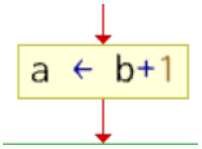
Como dijimos, en una variable se puede almacenar datos y como su nombre lo indica, su contenido puede variar a lo largo de la ejecución del algoritmo. En cambio, las constantes contienen información que no puede variar en ningún momento. Si el algoritmo utiliza valores constantes, éstos se deben declarar y posteriormente asignar el valor que va a contener. Una manera estándar de identificarlas es escribir su identificador en mayúsculas a lo largo de todo el algoritmo.

Instrucciones primitivas

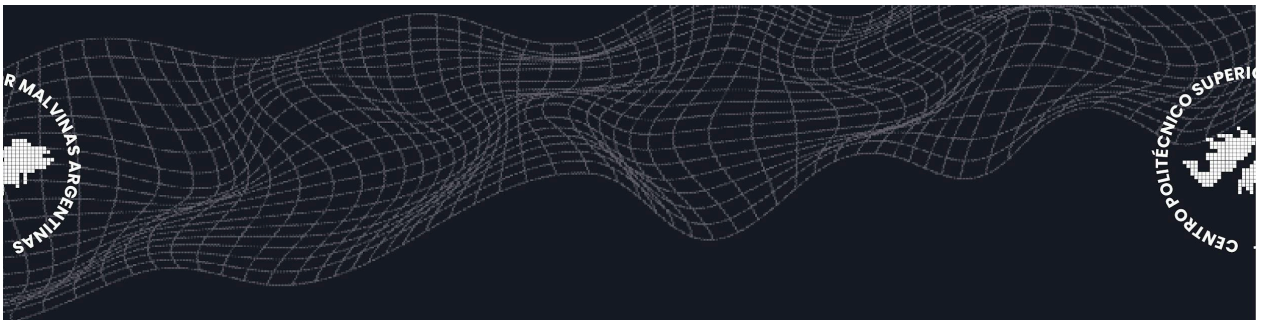
Son instrucciones que se ejecutan en cuanto son leídas. En ellas sólo puede haber:

- Asignaciones (←)
- Operación aritmética (+, -, * /,...)
- Identificadores (nombres de variables o constantes)
- Valores (números o texto encerrado entre comillas)
- Llamadas a subprogramas

En el pseudocódigo se escriben entre Algoritmos ... FinAlgoritmo. En los diagramas de flujo y tablas de decisión se escriben dentro de un rectángulo.



Acción de asignación en Diagrama de Flujo de Datos (DFD)



Instrucción de asignación

Permite almacenar un valor en una variable. Para asignar el valor se escribe el símbolo ←, de modo que:

identificador←valor

El identificador toma el valor indicado. Ejemplo:

```
a←8
```

Ahora a vale 8. Se puede utilizar otra variable en lugar de un valor. Ejemplo:

```
a←9
b←a
```

b vale ahora lo que vale a, es decir b vale 9. Los valores pueden ser:

- Números. Se escriben tal cual, el separador decimal suele ser el punto (aunque hay quien utiliza la coma).
- Caracteres simples. Los caracteres simples (un solo carácter) se escriben entre comillas simples: ‘a’, ‘c’,etc.
- Textos. Se escriben entre comillas doble “Hola” Lógicos. Sólo pueden valer verdadero o falso (se escriben tal cual)
- Identificadores. En cuyo caso se almacena el valor de la variable con dicho identificador.

En las instrucciones de asignación se pueden utilizar expresiones más complejas con ayuda de los operadores. Ejemplo:

```
b←(a*3)/2
```

Es decir, b vale el resultado de multiplicar el valor de a por tres y dividirlo entre dos.

Los Operadores Algebraicos permitidos son:

- + **Suma**
- **- Resta** o cambio de signo
- *** Producto**
- **/ División**

- **mod** Resto. Por ejemplo 9 mod 2 da como resultado 1
- **div** División entera. 9 div 2 da como resultado 4 (y no 4,5)
- ^ **Exponente**. 9^2 es 9 elevado a la 2

Hay que tener en cuenta la prioridad del operador. Por ejemplo, la multiplicación y la división tienen más prioridad que la suma o la resta. Sí *9+6/3* da como resultado 5 y no 11. Para modificar la prioridad de la instrucción se utilizan paréntesis. Por ejemplo *9+(6/3)*

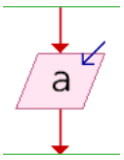
Instrucciones de entrada y salida

Lectura de datos

```
...
Leer a;
...
```

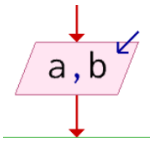
Es la instrucción que simula una lectura de datos desde el teclado. Se hace mediante la orden Leer y a continuación se indica el identificador de la variable que almacenará lo que se lea. Ejemplo (pseudocódigo):

El mismo ejemplo en un DFD sería:



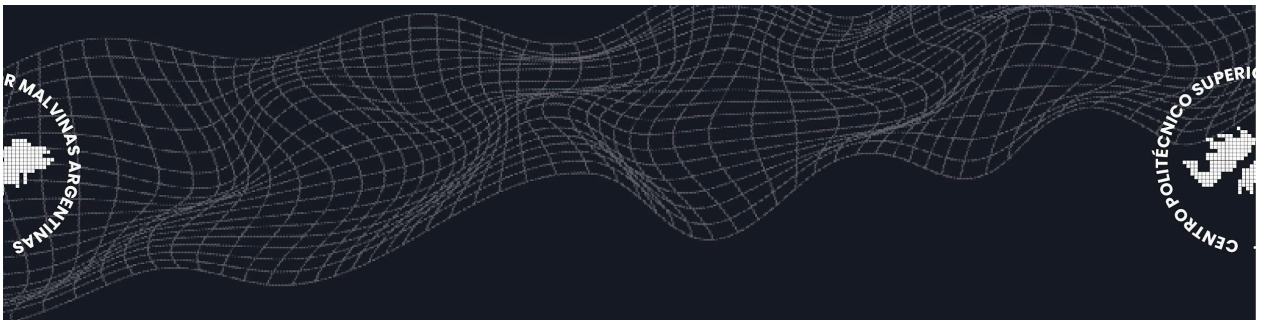
En ambos casos a contendrá el valor leído desde el teclado. Se pueden leer varias variables a la vez:

```
...
Leer a, b;
...
```



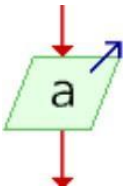
Escritura de datos

Funciona como la anterior pero usando la palabra Escribir. Simula la salida de



datos del algoritmo por pantalla.

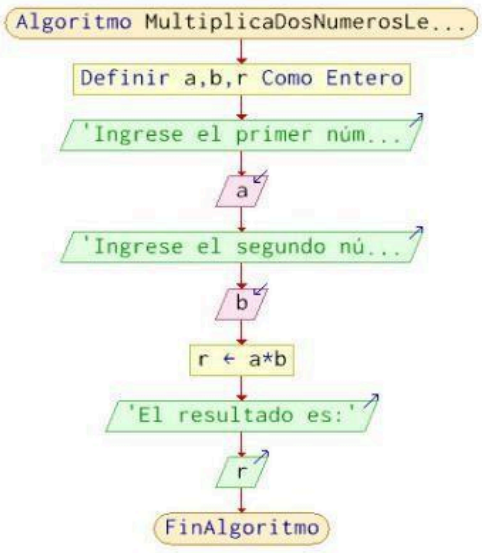
```
...
Escribir a;
...
```

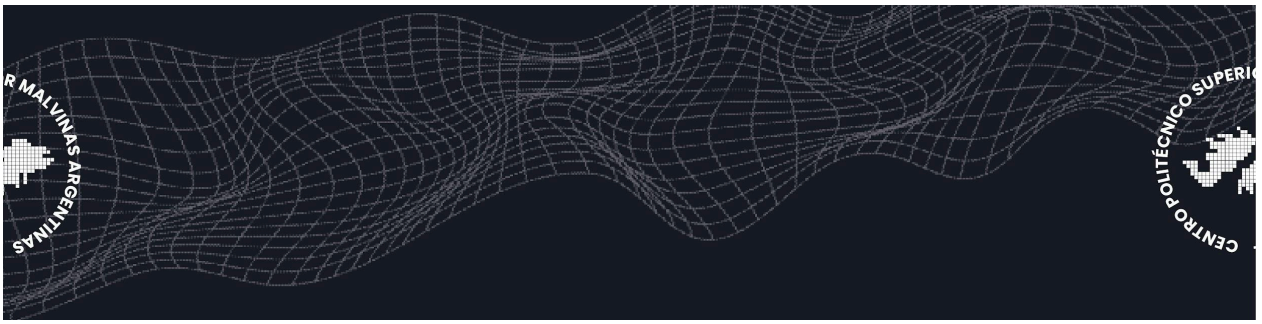


Ejemplo de algoritmo

El algoritmo completo que escribe el resultado de multiplicar dos números leídos por teclado sería (en pseudocódigo)

```
// Multiplicar dos números leídos por teclado.
Algoritmo MultiplicaDosNumerosLeidos
    Definir a,b,r Como Entero;
    Escribir 'Ingrese el primer número:';
    Leer a;
    Escribir 'Ingrese el segundo número:';
    Leer b;
    r ← a*b;
    Escribir 'El resultado es:';
    Escribir r;
FinAlgoritmo
```





Expresiones lógicas

Todas las instrucciones de este apartado utilizan expresiones lógicas. Son expresiones que dan como resultado un valor lógico (verdadero o falso).

OPERADOR	SIGNIFICADO	EJEMPLO
>	Mayor que...	`3>2`
<	Menor que...	`'ABC'<'abc'`
=	Igual que...	`4=3`
<=	Menor o igual que...	`'a'<='b'`
>=	Mayor o igual que...	`4>=5`
<>	Distinto que...	`'a'<>'b'`

También se pueden unir expresiones utilizando los operadores Y (en inglés AND), el operador O (en inglés OR) o el operador NO (en inglés NOT).

Estos Operadores Lógicos permiten unir expresiones lógicas. Por ejemplo:

EXPRESIÓN	RESULTADO VERDADERO SI...
`a>8 Y b<12`	La variable a es mayor que 8 y (a la vez) la variable b es menor que 12
`a>8 Y b<12`	O la variable a es mayor que 8 o la variable b es menor que 12. Basta que se cumpla una de las dos expresiones.
`a>30 Y a<45`	La variable a está entre 31 y 44
`a<30 O a>45`	La variable a no está entre 30 y 45
`NO a=45`	La variable a no vale 45
`a >8 Y NO b<7`	La variable a es mayor que 8 y b es menor o igual que 7
`a>45 Y a<30`	Nunca es verdadero

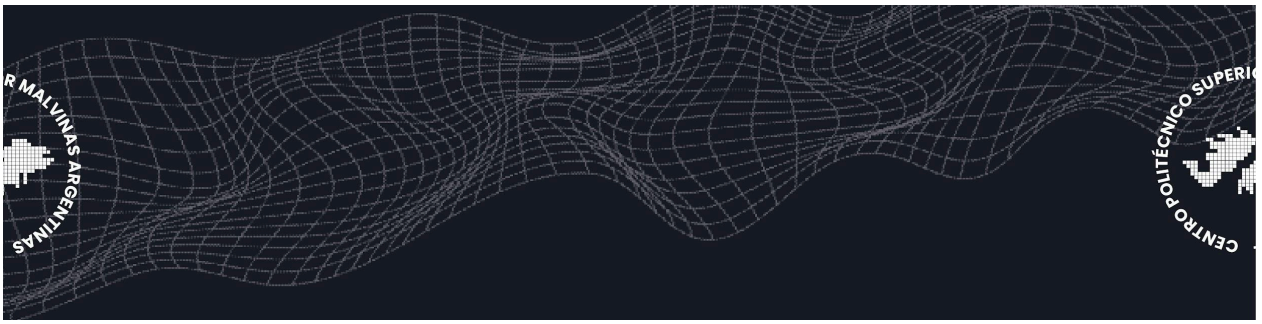
Te invitamos a ejercitar lo aprendido en la siguiente página. Es un test de aprendizaje la cual no tiene puntuación pero sirve para comprobar si entendió sobre expresiones lógicas:
https://www.mclibre.org/consultar/python/ejercicios/ej-booleanos.html

---

Instrucción de alternativa simple (Decisión simple)

La alternativa simple se crea con la instrucción si (en inglés if). Esta instrucción evalúa una determinada expresión lógica y dependiendo de si esa expresión es verdadera o no se ejecutan las instrucciones siguientes. Esto es equivalente al siguiente diagrama de flujo:

Las instrucciones sólo se ejecutarán si la expresión evaluada es verdadera.



Instrucción de alternativa doble

Se trata de una variante de la alternativa en la que se ejecutan unas instrucciones si la expresión evaluada es verdadera y otras si es falsa.

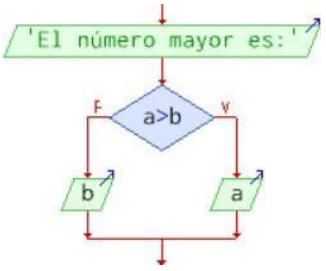
```
...
Si <condición> Entonces
<instrucciones>;
SiNo
<instrucciones>;
FinSi
...
```

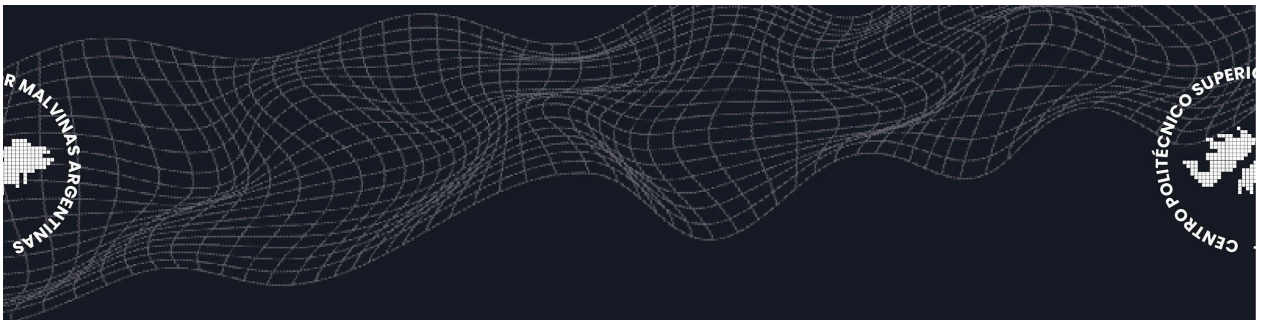
Funcionamiento

```
...
Escribir 'El número mayor es:';
Si a > b Entonces
```

```
//se ejecutan si la expresión es verdadero
Escribir 'El resultado es:';
SiNo
//se ejecutan si la expresión es falsa
Escribir b;
FinSi
...
```

Sólo se ejecutan unas instrucciones dependiendo de si la expresión es verdadera. El diagrama de flujo equivalente es:

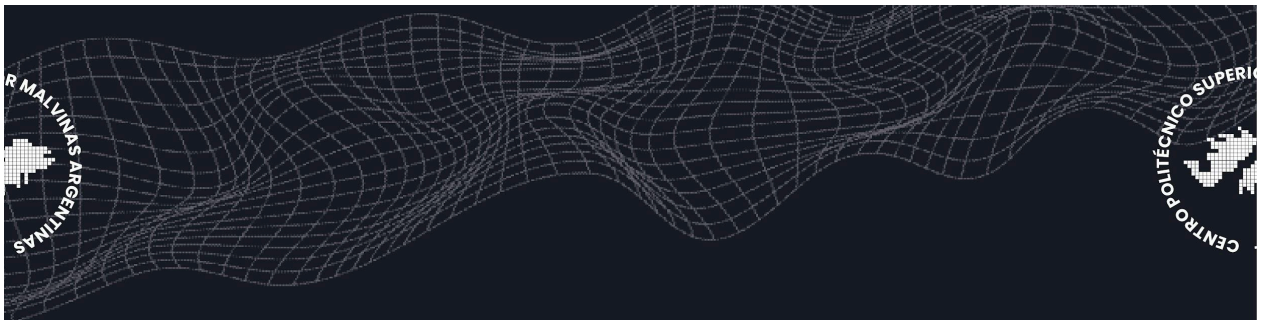




Hay que tener en cuenta que se puede colocar una instrucción si dentro de otro sí. A eso se le llama alternativas anidadas. Ejemplo:

```
...
Escribir 'El alumno alcanzó las competencias?';
Si nota>=6 Entonces
Si nota>=6 Y nota<=7 Entonces
Escribir 'SATISFACTORIAMENTE';
SiNo
Si nota>=8 Y nota<9 Entonces
Escribir 'MUY SATISFACTORIAMENTE';
SiNo
Escribir 'SOBRESALIENTEMENTE';
FinSi
```

```
FinSi
SiNo
Escribir 'NO ALCANZO';
FinSi
...
```



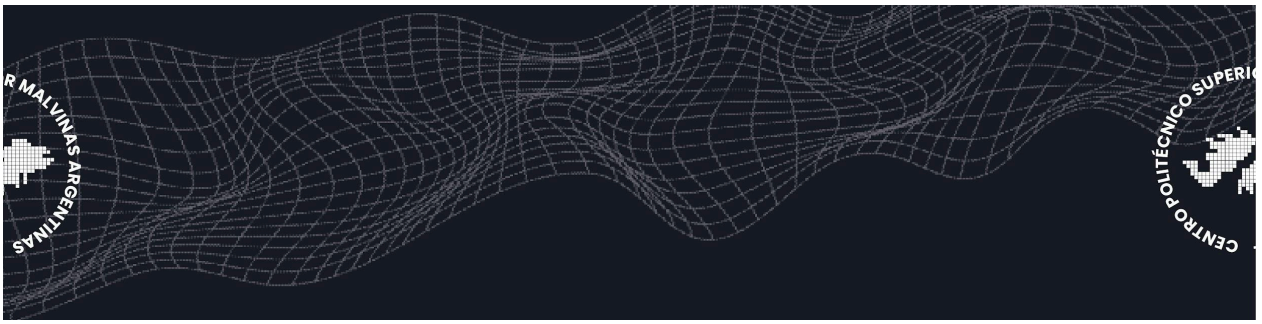
Instrucción de alternativa compuesta

```
      ...
Segun <variable> Hacer
<número1>: <instrucciones>;
    <número2>,<número3>: <instrucciones>;
    <...>
```

```
De Otro Modo:
<instrucciones>;
FinSegun
...
```

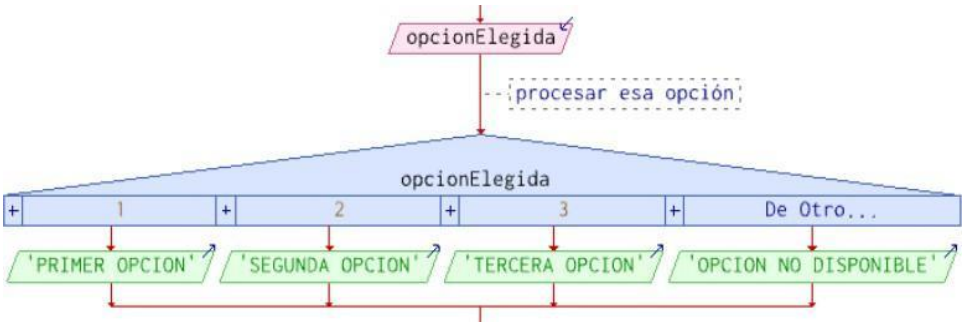
En muchas ocasiones se requieren condiciones que poseen más de una alternativa.
Ejemplo:

```
      ...
Escribir "Menú de opciones disponibles";
Escribir "  1. Primero";
Escribir "  2. Segundo";
Escribir "  3. Tercero";
Escribir "Elija una opción (1..3): ";
// ingresar una opcion
Leer opcionElegida;
// procesar esa opción
Segun opcionElegida Hacer
1:
Escribir "PRIMER OPCION";
2:
Escribir "SEGUNDA OPCION";
3:
Escribir "TERCERA OPCION";
De Otro Modo:
Escribir "OPCION NO DISPONIBLE";
FinSegun
...
```



Casi todos los lenguajes de programación poseen esta instrucción que suele ser un case (aunque C, C++, Java y C# usan switch). Se evalúa la expresión y si es igual que uno de los valores interiores se ejecutan las instrucciones de ese valor. Si no cumple ningún valor se ejecutan las instrucciones dentro de la sección De Otro Modo.

En DFD sería:



Instrucciones iterativas: ciclo Mientras

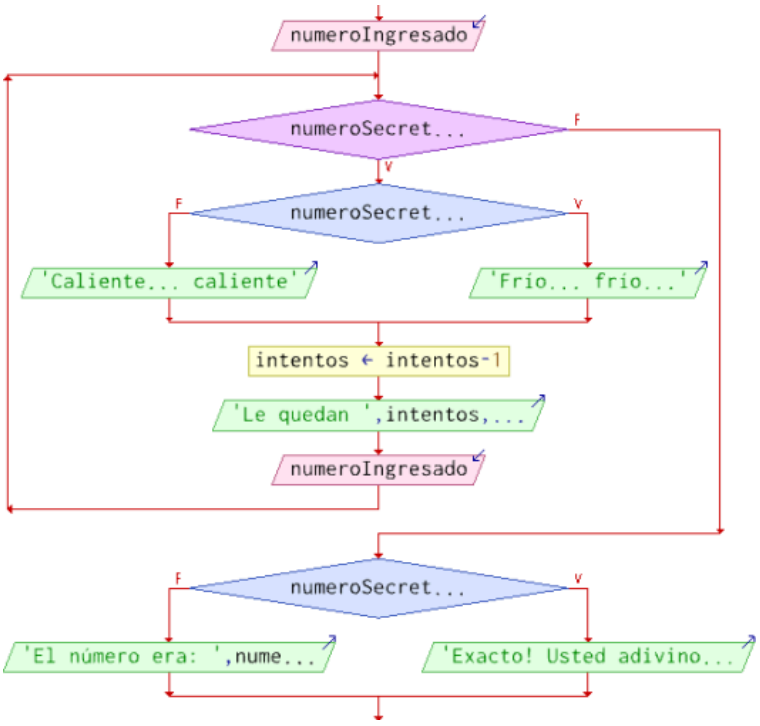
El pseudocódigo admite instrucciones iterativas. Las fundamentales se crean con una instrucción llamada mientras (en inglés while). Su estructura es:

```
...
Mientras <condición> Hacer
   <instrucciones>
FinMientras
```

Significa que las instrucciones del interior se ejecutan una y otra vez mientras la condición sea verdadera. Si la condición es falsa, las instrucciones se dejan de ejecutar.

```
...
Leer numeroIngresado;
Mientras numeroSecreto<>numeroIngresado & intentos>1 Hacer
    Si numeroSecreto>numeroIngresado Entonces
        Escribir "Frío... frío...";
    SiNo
        Escribir "Caliente... caliente";
    FinSi
    intentos = intentos-1;
    Escribir "Le quedan ",intentos," intentos:";
    Leer numeroIngresado;
FinMientras
...
```

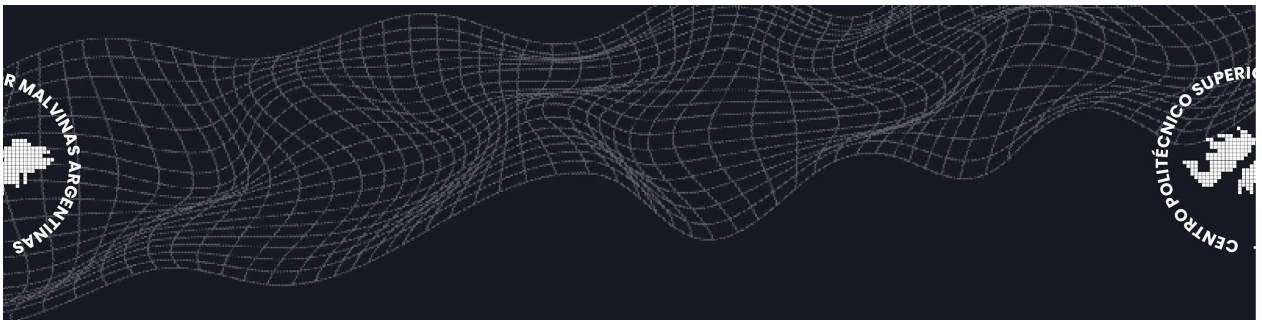
Es importante tener en cuenta que las instrucciones interiores a la palabra Mientras podrían incluso no ejecutarse si la condición es falsa inicialmente. En Diagrama de Flujo de Datos (DFD):



Instrucciones iterativas de tipo Repetir

Su función es muy similar a la instrucción Mientras. La diferencia con la anterior está en que se evalúa la condición al final (en lugar de al principio). Consiste en una serie de instrucciones que repiten continuamente su ejecución hasta que la condición sea verdadera (funciona por tanto al revés que el Mientras ya que, si la condición es falsa, las instrucciones se siguen ejecutando. Estructura

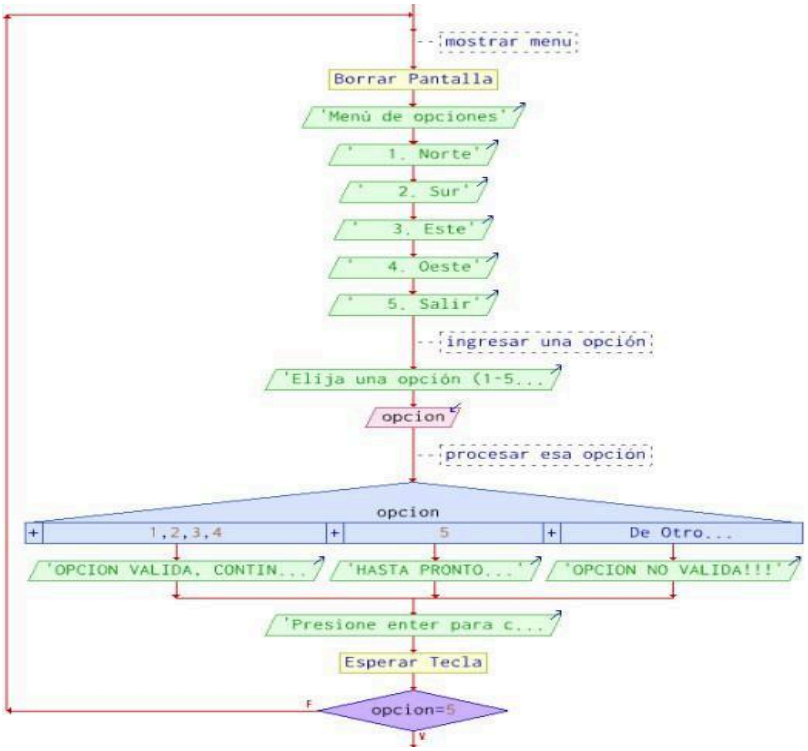
```
...
Repetir
    <instrucciones>;
Hasta Que <condición>
...
```



Ejemplo:

```
      <condición>...
      Repetir
          // mostrar menu
          Limpiar Pantalla;
          Escribir "Menú de opciones";
          Escribir " 1. Norte";
          Escribir " 2. Sur";
          Escribir " 3. Este";
          Escribir " 4. Oeste";
          Escribir " 5. Salir";
          // ingresar una opción
          Escribir "Elija una opción (1-5): ";
          Leer opcion;
          // procesar esa opción
          Segun opcion Hacer
              1,2,3,4:
                  Escribir "OPCION VALIDA, CONTINUA...";
              5:
                  Escribir "HASTA PRONTO...";
              De otro modo:
                  Escribir "OPCION NO VALIDA!!!";
          FinSegun
          Escribir "Presione enter para continuar";
          Esperar Tecla;
      Hasta Que opcion=5
...
```

El DFD equivalente es:



Instrucciones iterativas Para

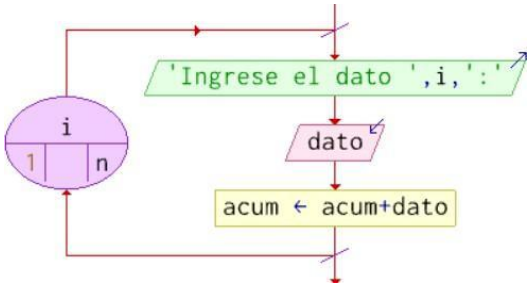
Existe otro tipo de estructura iterativa. En realidad, no sería necesaria ya que lo que hace esta instrucción lo puede hacer una instrucción Mientras, pero facilita el uso de bucles con contador. Es decir, son instrucciones que se repiten continuamente según los valores de un contador al que se le pone un valor de inicio, un valor final y el incremento que realiza en cada iteración (el incremento es opcional, si no se indica se entiende que es de uno).

```
    ...
    Para <variable> <- <inicial> Hasta <final> Con Paso <paso> Hacer
      <instrucciones>;
    FinPara
...
```

Ejemplo:

```
...
Para i=1 Hasta n Hacer
    Escribir "Ingrese el dato ",i,":";
    Leer dato;
    acum<-acum+dato;
FinPara
...
```

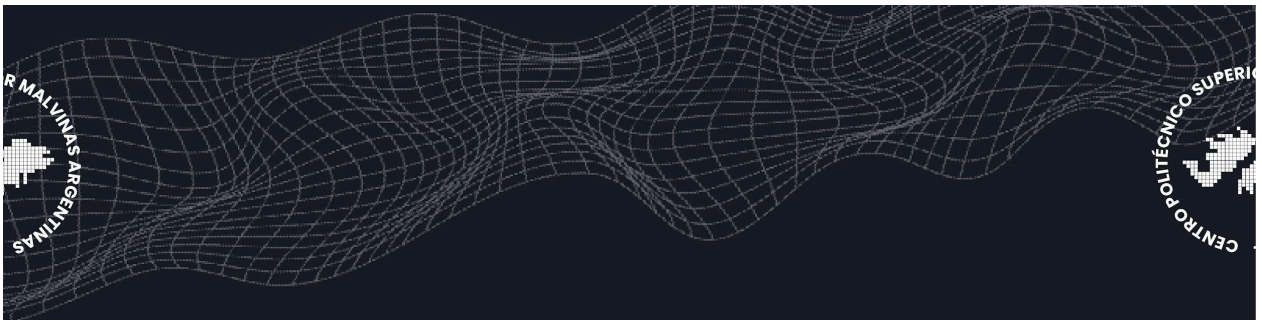
El diagrama de flujo equivalente a una estructura para sería:



Otros formatos de pseudocódigo utilizan la palabra Desde en lugar de la palabra Para (que es la traducción de for, nombre que se da en el original inglés a este tipo de instrucción).

Estructuras iterativas anidadas

Al igual que ocurría con las instrucciones si, también se puede insertar una estructura iterativa dentro de otra; pero en las mismas condiciones que la instrucción si. Cuando una estructura iterativa está dentro de otra se debe cerrar la iteración interior antes de cerrar la exterior (véase la instrucción si).



Prueba de Escritorio

Una prueba de escritorio es un tipo de prueba algorítmica, que consiste en la validación y verificación del algoritmo a través de la ejecución de las sentencias que lo componen (proceso) para determinar sus resultados

(salida) a partir de un conjunto determinado de elementos (entrada).

Qué mejor para explicar en qué consiste una prueba de escritorio que un buen ejemplo. Supongamos que deseamos implementar el algoritmo de Euclides. Este algoritmo es un método para encontrar el mayor común divisor entre dos números. El mayor común divisor es el número mayor por el que pueden dividirse ambos números dando como resultado un número entero sin resto. Una prueba de escritorio para el algoritmo de Euclides que obtiene el MCD de 15 y 4 consistiría, en primer lugar, la identificación de las variables declaradas:

```
a
b
resto
```

Se asignan los valores de las variables a y b respectivamente, que en base a lo que se desea saber sería:

```
a=15
b=4
```

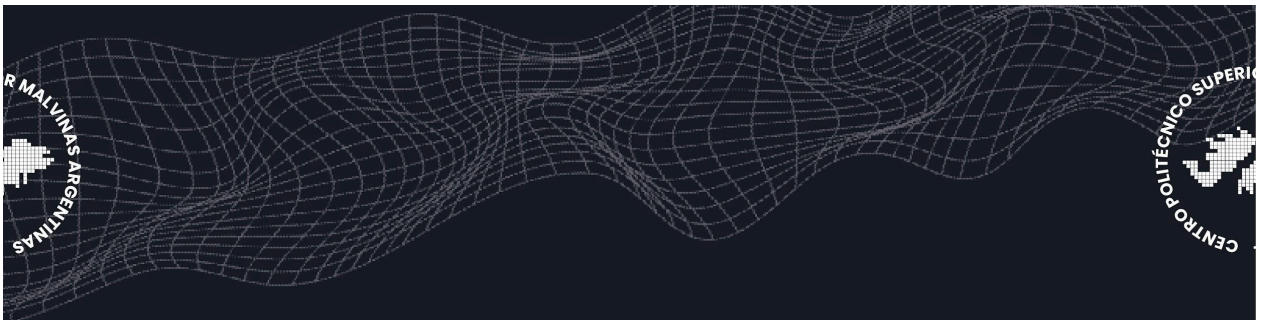
mientras que resto se define en base a la expresión resto <- a mod b, es decir:

```
a=15
b=4
resto=3
```

el siguiente paso es verificar la condición del Mientras, y como dicha condición es verdadera, se tendría la siguiente secuencia de valores para las variables, la cual es generada dentro del ciclo:

```
a=4, 3, 1
resto=3, 1, 0
```

de donde puede verse, que el MCD de 15 y 4 es 1.



Actividad

Debemos asegurarnos de comprender el proceso descrito y repetirlo. También podremos utilizar los mismos valores, pero intercambiados, para averiguar ¿qué sucede? ¿qué secuencia de valores se genera? ¿cuál es el MCD? ¿cuál es ahora el MCD de 15 y 10? ¿cuál es el MCD de 10 y 15?

Pruebe con otros valores hasta que comprenda el funcionamiento del algoritmo, y compruebe los resultados de sus pruebas de escritorio con los algoritmos descritos en diagrama de flujo y pseudocódigo respectivamente.

---

3. Cierre

Finalmente, en esta clase pudimos conocer las actividades del ciclo de vida clásico del desarrollo. Y describimos cada uno de sus pasos: análisis, diseño, codificación, ejecución, prueba y mantenimiento. También desarrollaremos las formas de representación estándar para el diseño de algoritmos: Diagramas de flujo, pseudocódigo. Como así también, explicaremos la Creación de Algoritmos: instrucciones, instrucciones de declaración, instrucciones primitivas, instrucciones de entrada y salida, instrucciones de control. Y finalmente hablaremos sobre la prueba de escritorio, para asegurarnos que el algoritmo creado funcione según lo esperado.

4. Bibliografía

● FUNDAMENTOS DE PROGRAMACIÓN. Algoritmos, estructura de datos y objetosCuarta edición - Luis Joyanes Aguilar - McGraw-Hill - 2008 - ISBN 978-84-481-6111-8 Bibliografía sugerida de la Unidad:

● Programación y Algoritmos - Aprenda a Programar en C y Pascal: Manuales Users - Maximiliano Bonanata - M.P. Ediciones - 2003 - ISBN 9875261564.

● La biblia del programador, Implementación y debugging: Manuales Users .code - Dante Cantone - M.P. Ediciones - 2003 - ISBN 9872299579