

UNIDAD DIDÁCTICA 1

IDENTIFICACIÓN DE LOS ELEMENTOS DE UN PROGRAMA INFORMÁTICO

MÓDULO PROFESIONAL:
PROGRAMACIÓN



CESUR
Tu Centro Oficial de FP

Índice

RESUMEN INTRODUCTORIO	2
INTRODUCCIÓN	2
CASO INTRODUCTORIO	3
1. VISIÓN HISTÓRICA	4
2. LA PROGRAMACIÓN DE ORDENADORES	5
3. CONCEPTO DE ALGORITMO.....	7
4. LENGUAJES DE PROGRAMACIÓN	12
5. DATOS Y TIPOS DE DATOS	16
6. CONSTANTES Y VARIABLES	18
7. EXPRESIONES	20
8. OPERACIONES DE ASIGNACIÓN.....	22
9. REPRESENTACIÓN DE ALGORITMOS. SOFTWARE DE UTILIDAD.....	23
10. ESTRUCTURAS DE CONTROL.....	47
10.1 Estructuras selectivas o alternativas	47
10.2 Estructuras repetitivas	57
RESUMEN FINAL	70

RESUMEN INTRODUCTORIO

A lo largo de esta unidad veremos una visión histórica de la informática y la programación de los ordenadores. También explicaremos el concepto de algoritmo y cómo se pueden representar, conoceremos los distintos lenguajes de programación, los distintos datos existentes y sus tipos, la definición de variable y constante. Seguiremos viendo las expresiones y los distintos tipos de operadores existentes.

Por último, estudiaremos la forma de escribir pseudocódigo dentro de la aplicación PseInt, las diferentes estructuras de control que se pueden utilizar y aprenderemos lógica de programación antes de comenzar de lleno con la programación de aplicaciones en lenguaje de programación Java que se tratará a partir de la siguiente unidad.

INTRODUCCIÓN

En la actualidad todo el mundo utiliza la tecnología para casi todos los ámbitos de sus vidas. El software está presente no solo en los ordenadores, sino en los dispositivos móviles en los cuales almacenamos gran parte de aquella información que necesitamos prácticamente a diario: números de teléfono, número de la cuenta del banco, avisos para las tareas que tenemos que realizar día a día, las fechas de los cumpleaños de las personas más cercanas a nosotros, podemos acceder a un mundo de información a través de las conexiones a Internet, etc.

Realmente tenemos dos opciones: podemos ser simplemente consumidores de todo ese software o podemos ser una de las personas que crea esos programas. Los programadores pueden dividir un problema grande en problemas más pequeños. Eso es pensar de manera diferente, tal y como expresaba Steve Jobs. Hay muchas personas que de forma autodidacta aprenden algún lenguaje de programación para hacer sus propios programas.

La tarea de sentarse delante del ordenador para realizar un programa requiere además de mucha paciencia, unos conocimientos básicos que le permitan saber que lo que se está creando va a funcionar. Aprender a programar puede ser fácil si empezamos desde lo más básico y se va evolucionando de forma gradual hasta que somos capaces de realizar un programa bien construido y que funciona. La programación se aprende **practicando**, como se aprende a montar en bicicleta o a realizar experimentos en un laboratorio o a resolver problemas de matemáticas. Los primeros pasos consisten en aprender a **escribir un algoritmo**, **elegir un lenguaje de programación** que se adapte a nuestras necesidades y traducir ese algoritmo a ese lenguaje. Una vez que sepamos

programar en un lenguaje, el hecho de aprender otros lenguajes de programación supone una tarea mucho más fácil que empezar desde cero.

El flujo de un programa es el orden en el que se ejecutan las sentencias que forman parte del cuerpo de un programa. Las estructuras de control son una característica básica de los lenguajes que se utilizan para modificar el flujo de un programa. Estas estructuras permiten realizar diferentes tareas con la información y son las que aportan diversas posibilidades en la programación.

Aunque nos parezca increíble las estructuras de control están presentes en nuestro día a día. ¿Cuántas veces no nos habrán dicho: si está lloviendo coge el paraguas? Y nunca nos hemos parado a pensar que se trata de una estructura de control en concreto una estructura selectiva simple (SI).

CASO INTRODUCTORIO

Trabajas en una empresa de desarrollo de software que está desarrollando un proyecto. Estás en la fase de codificación y lo primero en lo que piensas es en teclear el código directamente en el lenguaje de programación elegido, pero antes debes realizar una planificación y centrarte en los requisitos del proyecto recopilando información sobre aquellas funcionalidades, objetivos y limitaciones tanto de software como técnicas del proyecto y la empresa de la que formas parte.

Al finalizar el estudio de la unidad serás capaz de realizar dado un problema, el algoritmo que lo resuelve, conocerás los distintos métodos de representación de algoritmos, utilizará los diagramas de flujo, diagramas N-S y pseudocódigo para realizar el paso previo al desarrollo del programa en un lenguaje de programación y aprenderás a usar y combinar diferentes estructuras de control en pseudocódigo para resolver un problema dado.

1. VISIÓN HISTÓRICA

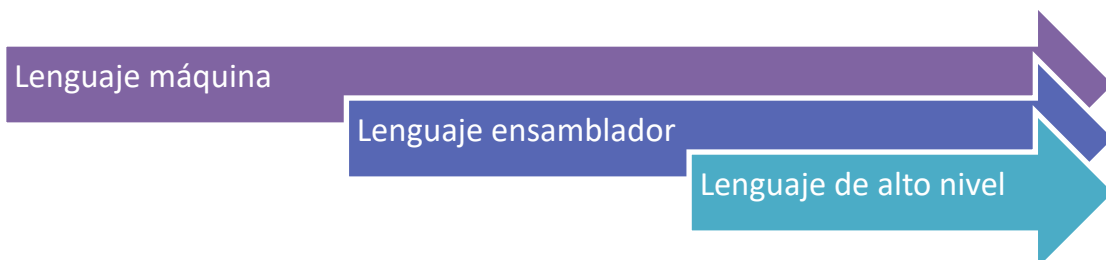
Para el nuevo proyecto asignado, te dejan escoger el lenguaje de programación, siempre y cuando sea un lenguaje orientado a objetos. Por tanto, crees conveniente repasar los tipos de lenguajes que han surgido y como han evolucionado desde que comenzaron a utilizarse los ordenadores.

La **Informática** se define como la ciencia del tratamiento automático y racional de la información, así como el soporte del conocimiento y las comunicaciones. Al adentrarnos en el estudio del mundo de la informática se puede comprobar que existe un amplio abanico de áreas y especialidades. Por ello, antes de empezar a estudiar cualquiera de ellas, es necesario asentar unas bases o pilares sobre los que sustentar los futuros conocimientos que se vayan adquiriendo.

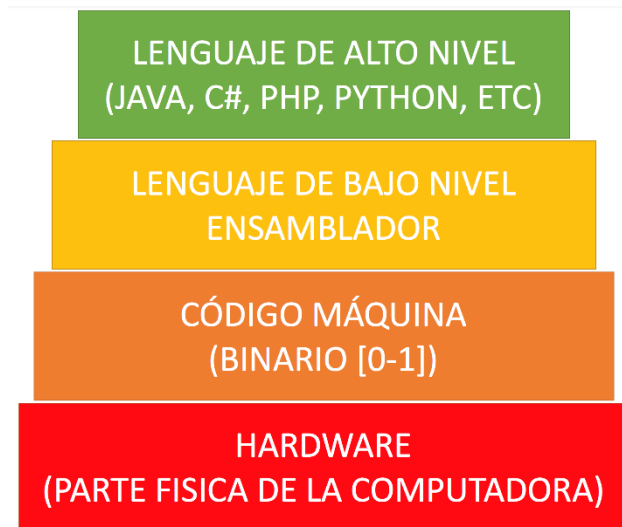
De la misma forma que el comportamiento y el pensamiento humano se rigen por métodos de razonamiento lógicos que nos permiten la ejecución de acciones o tareas concretas, el comportamiento o actuación de un ordenador se rige por lo que denominamos programación, entendiendo como tal el planteamiento, desarrollo y puesta en marcha de soluciones a problemas concretos, mediante una secuencia de instrucciones o conjunto de acciones lógicas que debe ejecutar el ordenador y que son transmitidas a éste por la figura del programador en forma de programa.

Cuando se empezaron a diseñar las primeras computadoras, el único lenguaje de programación del que se disponía era el lenguaje máquina. Estos lenguajes eran difíciles de leer y modificar, por lo que se desarrollaron lenguajes ensambladores, que facilitaban el trabajo a los programadores. Dichos lenguajes están definidos por un código nemotécnico que establece una cierta equivalencia entre las instrucciones del código máquina y las instrucciones del lenguaje ensamblador.

Posteriormente aparecieron los lenguajes de programación de alto nivel (como Pascal o C), con los que el programador puede escribir un programa aun sin conocer extensivamente la máquina en que dicho programa va a ser ejecutado. El lenguaje en estos casos es independiente de la máquina.



Un lenguaje de alto nivel está más cercano a los lenguajes naturales (generalmente el inglés), teniendo así menos limitaciones.



Clasificación lenguajes de programación.

Fuente: <https://conogasi.org/articulos/lenguaje-de-programacion/>

2. LA PROGRAMACIÓN DE ORDENADORES

Te ha gustado mucho todo lo que has aprendido sobre la informática y lenguajes de programación y te gusta cada vez más la idea de desarrollar el proyecto en lenguaje Java usando programación estructurada. Repasas de nuevo los requisitos del proyecto y los plazos de entrega, no te puedes demorar mucho es hora de tomar decisiones.

Los ordenadores no son inteligentes por sí mismos, todo lo que hacen es porque alguien le ha indicado qué tiene que hacer. La forma de comunicar a un ordenador qué debe hacer es, generalmente, mediante el uso de programas que contienen las instrucciones e información necesaria para resolver el problema planteado.



CITA

“Se suele decir que una persona no entiende algo de verdad hasta que puede explicárselo a otro. En realidad, no lo entiende de verdad hasta que puede explicárselo a un computador”. Donald Knuth.

Para comunicarle al ordenador las instrucciones a ejecutar se debe utilizar un lenguaje que pueda entender. Existen muchos lenguajes de programación, pero todos ellos tienen en común una serie de normas semánticas y sintácticas.

Con el objeto de facilitar el planteamiento, desarrollo y puesta en marcha de los programas, surge la **metodología de la programación**, en sus dos vertientes más destacadas, la metodología estructurada y modular, también conocida como metodología tradicional y por otro lado la metodología orientada a objetos, que es un enfoque mas avanzado y que se basa en el concepto de “objetos” que interactúan entre sí, permitiendo la **encapsulación** (agrupación de datos y métodos en una sola entidad llamada clase y ocultación de la misma a otras partes del código), el **polimorfismo** (capacidad de un objeto para comportarse de diferentes formas según el contexto), **modularidad** (que favorece el desarrollo y mantenimiento de forma independiente, así como la reutilización de código) y la **herencia** (creación de nuevas clases basadas en otras) entre otras ventajas.

La esencia de la metodología de la programación se basa en el empleo de técnicas de **programación estructurada**, que consisten en:

1. Desarrollar un programa en módulos con un diseño descendente. (descomponer el problema en partes más pequeñas).
2. Emplear únicamente tres clases de estructuras básicas de control (secuencial, alternativa y repetitiva) en el desarrollo de cada módulo.

Con el objeto de facilitar el desarrollo, depuración y futuro mantenimiento de los programas, permitiendo el uso de parte de los programas ya existentes, a finales de la década de los 50 y comienzo de los 60 se desarrollaron los denominados lenguajes de programación con técnicas estructuradas.

Años después, en la década de los años 80 y utilizando como pilar la programación estructurada, surge la **programación orientada a objetos**, con nuevos lenguajes de programación (**Lenguajes Orientados a Objetos**) que dotan al programador de nuevos elementos para el análisis y desarrollo del software, aportando un nuevo enfoque al mundo de la programación.

En la actualidad se tiene, por otra parte, el concepto de desarrollo web. Este tipo de desarrollo utiliza básicamente lenguajes de programación (PHP, JSP, ASP.NET...), lenguajes de marcas (HTML) y bases de datos (MySQL, Oracle, SQL Server...), para la creación de páginas web. Para conseguir todo esto se usan tecnologías de programación del lado del cliente (para desarrollo del front-end) y del lado del servidor (para desarrollo del back-end).



ENLACE DE INTERÉS

Se puede ampliar información sobre la evolución de los lenguajes de programación, así como sus principales características:

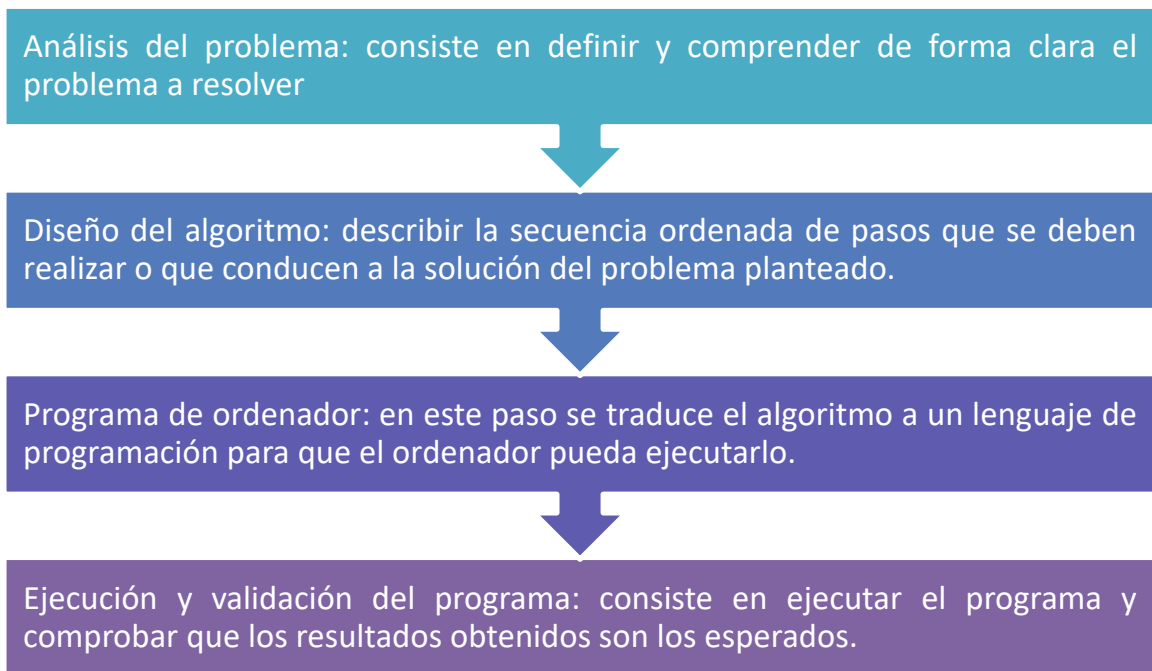


3. CONCEPTO DE ALGORITMO

Un amigo tuyo que es Software developer y que tiene más experiencia que tú, te ha recomendado empezar a diseñar los algoritmos de las funcionalidades más importantes del proyecto, para tener una visión más cercana de lo que debe hacer el sistema, de los datos que maneja y la forma de tratarlos.

Un algoritmo es un conjunto de acciones u operaciones a realizar por el ordenador, de forma clara y detallada, así como el orden en que deben ejecutarse, que nos conducen a la solución del problema, facilitando así su traducción posterior al lenguaje de programación correspondiente.

La resolución de todo problema exige el diseño de un algoritmo, y los pasos para dicha resolución son:

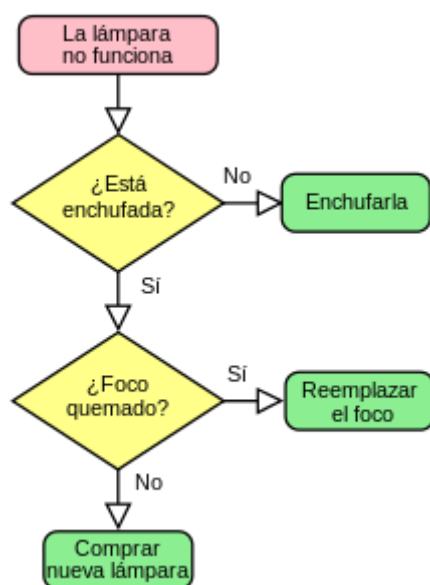


Los algoritmos son independientes, tanto del lenguaje de programación al que serán traducidos como del ordenador en que se ejecutarán.

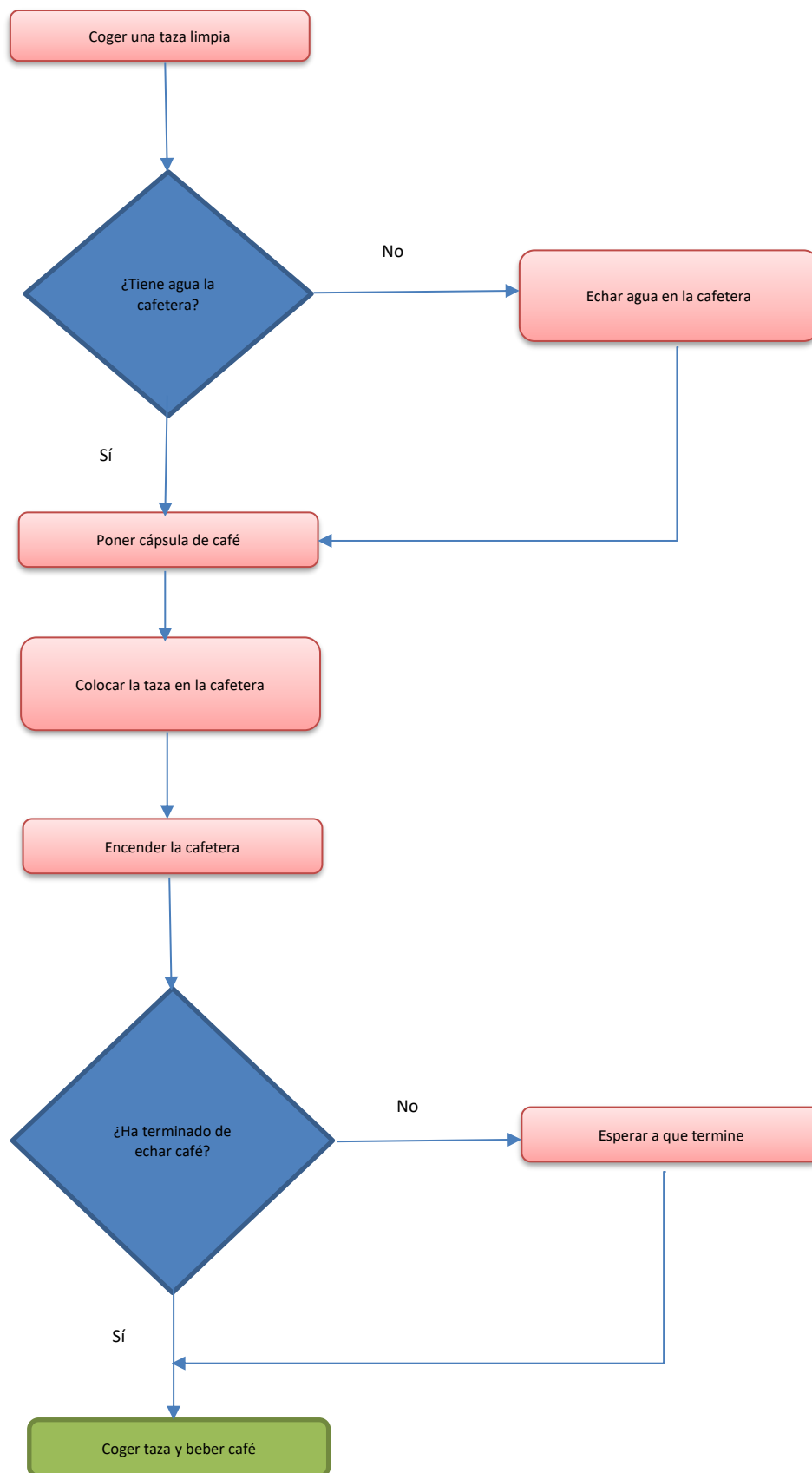
Características del algoritmo

Las características fundamentales que debe cumplir todo algoritmo son:

1. Debe ser conciso y detallado: reflejar *al máximo detalle* el orden de ejecución de cada acción que vaya a realizar el ordenador.
2. Debe ser exacto o preciso: si se sigue el mismo número de veces con los mismos datos de entrada, se deben obtener *siempre los mismos resultados*.
3. Debe ser finito o limitado: debe terminar en algún momento.
4. No debe ser rígido en su diseño: debe ser flexible a las diferentes representaciones gráficas y facilitar las modificaciones o actualizaciones del diseño realizado.
5. Debe ser claro y sencillo: con instrucciones fáciles de entender, evitando complejidades innecesarias y con una estructura lógica y coherente que facilite el entendimiento.



Ejemplo algoritmo para resolver un problema de la vida diaria.



Ejemplo algoritmo para resolver un problema de la vida diaria: preparar un café.



ARTÍCULO DE INTERÉS

Lee esta interesante historia de Ada Lovelace, considerada como la primera programadora de computadoras:



ARTÍCULO DE INTERÉS

Consulta este artículo sobre algoritmos y los diferentes tipos de algoritmos que puedes encontrar:



ENLACE DE INTERÉS

En este enlace podrás obtener diversos cursos sobre algoritmia:





VÍDEO DE INTERÉS

En este vídeo encontrará una explicación detallada sobre el concepto de algoritmo:



4. LENGUAJES DE PROGRAMACIÓN

Ya has decidido que el proyecto será implementado en Java. Por lo cual, serán necesarias una serie de herramientas software que tendrás que instalar para preparar el entorno donde empezar a desarrollar la aplicación. Una de las herramientas que necesitas tener instalada es la JVM que será la encargada de realizar la traducción a código máquina.

Una vez definido, el algoritmo debe ser suministrado al procesador, el cual debe ser capaz de interpretarlo, lo que significa que debe comprender las instrucciones de cada paso y realizar las operaciones correspondientes.

Cuando el procesador es un ordenador, el algoritmo deberá ser expresado en un formato denominado programa, que se escribe en un lenguaje de programación.

Un lenguaje de programación es una colección de símbolos y caracteres combinados entre sí con una sintaxis ya definida, para permitir transmitir instrucciones a la computadora.

Existen 3 tipos principales de lenguajes de programación en esta unidad los clasificaremos de la siguiente forma:

Lenguaje máquina o binario:

- Aquellos que están escritos en *lenguajes directamente inteligibles por la máquina*. Es decir: mediante cadenas binarias, que son secuencias de ceros y unos. Un inconveniente es que el lenguaje máquina depende del hardware. También es un lenguaje de bajo nivel, pero para diferenciarlo del lenguaje ensamblador se le denomina lenguaje máquina o binario.

- Ejemplo de instrucción: 01001001 10010010 11000011 00001111

Lenguaje de bajo nivel o ensamblador:

- Son más fáciles de usar que los lenguajes máquina, pero, al igual que el lenguaje máquina, depende de la máquina en particular. Al programar en bajo nivel, el programa tiene que ser traducido a binario para que lo entienda la máquina. El programa de bajo nivel se llama “programa fuente” y el traducido al binario “programa objeto”.
- Ejemplo de instrucción: MOV AX, 5 ; Mueve el valor 5 al registro AX
- Ejemplo de instrucción: ADD AX, 3 ; Suma el valor 3 al registro AX

Lenguaje de alto nivel:

- Son los que se utilizan a día de hoy. Por sus características se encuentran más próximos al usuario o programador. Una de las características más importantes es que son independientes de la arquitectura del ordenador. Estos lenguajes no se pueden ejecutar directamente en el ordenador, tienen que ser traducidos o compilados a lenguaje máquina. También tenemos “programa fuente” en alto nivel y se traduce a “programa objeto” por medio de dos tipos de programas: compiladores e intérpretes. La principal diferencia entre un compilador y un intérprete es que el intérprete acepta un programa fuente que traduce y ejecuta simultáneamente analizando cada instrucción por separado, y el compilador efectúa dicha operación en dos fases independientes: primero traduce todo y a continuación lo ejecuta.
- Ejemplo de instrucción: int x = 5;

Una de las primeras y principales distinciones que se pueden hacer al clasificar los lenguajes de programación es si el lenguaje es **compilado** o **interpretado**.

Un lenguaje de programación se dice que es **compilado** cuando existe un traductor que recoge el programa/aplicación realizada por un programador y lo compila para un determinado sistema operativo, es decir lo convierte en un ejecutable que un sistema operativo o arquitectura determinada entiende y puede ejecutar. (es decir...primero se traduce todo y a continuación se ejecuta).

Ejemplos de lenguajes de programación compilados son: C, C++, C#, Swift, Objective-C...

Los lenguajes **interpretados**, tal y como indica la palabra, son lenguajes de programación que necesitan de un intérprete para ser ejecutados. Ejemplos de estos lenguajes son el lenguaje PHP, JavaScript, Ruby...

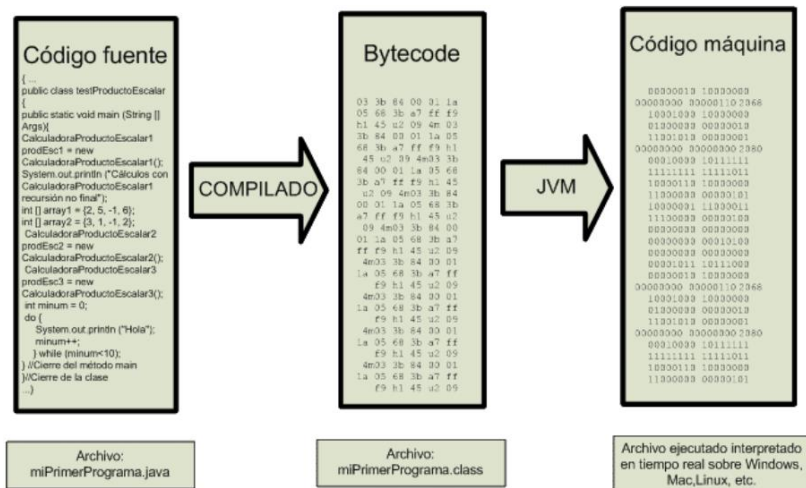
El caso del lenguaje **Java** es un caso especial, ya que se encuentra entre lo que es un lenguaje compilado y uno interpretado. Java no realiza la compilación pura como lo realizan el resto de los lenguajes compilados, sino que realiza un paso intermedio que consiste en que el compilador traduce el código de alto nivel a un código intermedio denominado **bytecode**. Este código intermedio es posible distribuirlo a cualquier plataforma, pero para poder ser ejecutado, el ordenador destino deberá tener instalado un ejecutor denominado **Java Virtual Machine(JVM)**, el cual sí que es dependiente del sistema operativo y por lo tanto del hardware.

En la actualidad los lenguajes de programación más populares son: Java, C, C++, Python, Visual Basic .NET, C#, PHP, JavaScript, SQL, Objective-C, Swift, Ruby y Kotlin.



Ejemplo de Compilador e interprete.

Fuente: <https://marielinformaticauat.blogspot.com/p/ensambladores-compiladores-e-interpretes.html>



Ejemplo de compilación/traducción en JAVA.

Fuente: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=392:la-maquina-virtual-java-jvm-o-java-virtual-machine-compilador-e-interprete-bytecode-cu00611b&catid=68&Itemid=188



VÍDEO DE INTERÉS

Para conocer más sobre las diferencias entre lenguajes de programación, consulta este vídeo.



5. DATOS Y TIPOS DE DATOS

Te están empezando a surgir algunas dudas sobre los datos que vas a utilizar para almacenar la información de entrada y salida. La forma de almacenar los datos y el tipo empleado es muy importante establecerlo correctamente desde el principio, ya que una vez esté en marcha el desarrollo, cualquier cambio puede dar lugar a tener que reestructurar código produciendo retrasos en los plazos de entrega e incremento del coste del producto. Para evitarte estos inconvenientes realizas una tabla de datos o diccionario de datos donde recoges el nombre de identificador que almacena cada dato, el tipo de dato, la longitud y si es un tipo de datos simple o compuesto. De esta forma, a la hora de la codificación estará todo bien claro.

El primer objetivo de todo ordenador es el manejo de datos. Un dato es la expresión general que describe los objetos con los cuales opera un ordenador.

Una primera clasificación en base a la naturaleza de los datos es en datos de entrada y datos de salida.

En el contexto de la programación dato de entrada es toda aquella información que se introduce en un sistema informático para que sea procesada. Estos datos pueden provenir de diferentes fuentes de entrada como por ejemplo el teclado, archivos, bases de datos, lector de códigos e incluso otra aplicación informática. Por otro lado, datos de salida son los resultados o la información generada por el sistema informático después de procesar los datos de entrada.

Los datos de salida también pueden adoptar diferentes formas, a continuación, se indican algunas:

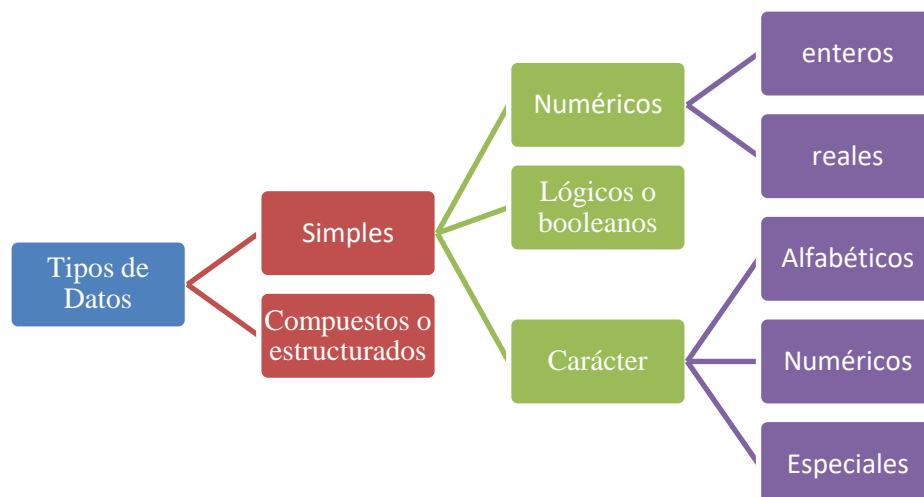
- **Salida por pantalla:** Los datos se muestran en pantalla bien como texto, gráficos, tablas, etc.
- **Archivos:** Se pueden guardar en archivos, ya sea en formato texto, imágenes, documentos, etc.
- **Bases de datos:** Los resultados se pueden almacenar en una base de datos para poder consultarlos cuando se necesiten.

Otra característica muy importante de los datos utilizados por un sistema informático es el tipo de dato que almacena o representa.

La mayoría de los ordenadores pueden trabajar con varios tipos de datos.

En general los tipos de datos básicos y más conocidos son los números enteros, números con decimales, datos de tipo booleano y de tipo carácter. Dependiendo de cada lenguaje de programación se ampliará el tipo de datos utilizado y se empleará una sintaxis determinada para representarlos.

- Los números **enteros** son números que no tienen coma, solo tienen parte entera.
- Los números **reales** tienen coma, es decir, tienen decimales por tanto se componen de parte entera y parte decimal.
- Un dato tipo **booleano** solo puede tener uno de dos valores permitidos que son verdadero(true) o falso(false) fundamental cuando se toman decisiones basadas en condiciones. Ej: Si alarma está activa o true entonces...realizar accion1.
- Carácter, que se emplean para representar letras, números o caracteres especiales que deben ir encerrados normalmente entre comillas simples.



Ejemplos de tipos de datos:

- Numéricos: 5, 25, 5963, 25,5 ó 25.5 (con números decimales se utiliza ,(coma) ó .(punto) dependiendo del lenguaje y la configuración del compilador).
- Lógicos o booleanos: true o false (Verdadero o falso).
- Carácter: 'a', '1', '@'.



VÍDEO DE INTERÉS

Amplia información sobre los tipos de datos en este enlace:



6. CONSTANTES Y VARIABLES

Estas avanzando mucho y ya tienes bien estructurada la información que forma parte de tu sistema. Pero tienes que diferenciar entre lo que serán variables y constantes. Sobre todo para que no haya ningún problema en cuanto a acceso a datos que no deban cambiar, es por ello, que todos los datos que son intocables y que solo son usados realizar operaciones consultas deberán ser declarados como constantes.

Los programas de ordenador contienen ciertos valores que no deben cambiar durante la ejecución del programa, estos valores se denominan **constantes**. Hay otros valores que cambian durante la ejecución del programa, para almacenar dichos valores se utilizan **variables**. En ambos casos lo que estamos haciendo es reservar un espacio en memoria en el cual se almacenarán el valor que tome la variable o el valor de la constante para cada uno de los identificadores definidos. Por tanto, declarar una variable significa indicar el tipo de dato que va a almacenar dicha variable, esto hay que hacerlo antes de poder usarla.

Lo normal es declarar las variables todas al inicio del algoritmo o programa, pero otras veces las iremos declarando según vayamos necesitando. También podemos agrupar variables que son de un mismo tipo en el momento de la declaración.

```
tipo_de_dato nombre o identificador;
```



EJEMPLO PRÁCTICO

En pseudocódigo, la forma más simple de declarar una variable es la siguiente:

```
Var contador: numérico , suma: numérico
```

Si tenemos muchas variables de un mismo tipo, podríamos agruparlas separándolas con comas de la forma siguiente:

```
Var contador, suma: numérico
```

En pseudocódigo no existe una sintaxis estándar, ya que puede variar del contexto y del autor. Pero una forma de definir una constante podría ser la siguiente:

```
CONSTANTE nombre_constante = valor
```

La palabra **CONSTANTE** es la palabra clave que indica que se está definiendo una constante, “nombre_constante” es el nombre que se le asigna a dicha constante y “valor” es el valor que se le asigna y que no se puede cambiar durante la ejecución.



EJEMPLO PRÁCTICO

En este ejemplo se asigna el valor 3.1416 a la constante con nombre PI. Recuerda que esta sintaxis puede variar dependiendo del pseudocódigo utilizado, de las convenciones y pautas establecidas en dicho contexto.

```
CONSTANTE PI = 3.1416
```

7. EXPRESIONES

Para realizar los algoritmos de cada una de las funcionalidades que requiere el sistema es necesario conocer como combinar las diferentes expresiones numéricas y lógicas esenciales para realizar cálculos y tomar decisiones en tus programas. Ya las conoces, pero repasas nuevamente para tenerlo todo claro antes de comenzar de lleno a programar en Java.

Una expresión es una combinación de constantes, variables, símbolos de operaciones, paréntesis y nombres de funciones especiales, de cuya evaluación se obtiene un único resultado. Dependiendo del resultado obtenido en la evaluación de una expresión se puede hablar de expresiones aritméticas y expresiones lógicas.

- **Expresión aritmética** es aquella compuesta por operaciones matemáticas:
 - Suma: +
 - Resta: -
 - Multiplicación: *
 - División: /
 - Resto: %, Mod
- **Expresión lógica:** el resultado de su evaluación es un valor lógico o booleano. Pueden usar operadores relacionales u operadores lógicos.

Relacionales

>, <, >=, <=, =, !=

Lógicos

Y lógico: &&, and

O lógico: ||, or

No ó negación: !, not

- Un **operador Y** devuelve V sólo cuando los 2 operandos sean verdaderos(V).

Operando 1	Operando 2	Resultado
V	V	V
V	F	F
F	V	F
F	F	F

- Un **operador O** devuelve V siempre que algún operando sea verdadero.

Operando 1	Operando 2	Resultado
V	V	V
V	F	V
F	V	V
F	F	F

- **Operador NO** es un operador unario (sólo 1 operando). Devuelve el valor opuesto al operador.

Operando	Resultado
V	F
F	V

Precedencia de operadores: Cuando una operación está compuesta por distintos tipos de operadores, para evaluarla hay que seguir un orden de prioridad:

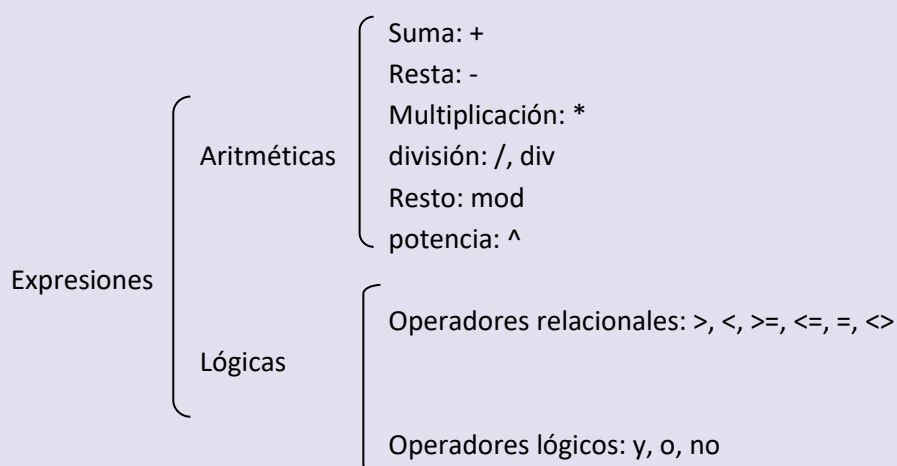
- 1º ()
- 2º ^ , no
- 3º * , / , div , mod , y
- 4º + , - , o
- 5º operaciones relacionales (> , < , >= , <= , = , !=)

Si todos los operadores tienen el mismo grado de prioridad se evalúa de izquierda a derecha.



RECUERDA

En la mayoría de los lenguajes de programación los operadores se representan de la siguiente forma:





ENLACE DE INTERÉS

Puede ampliar información sobre el uso de los operadores en este enlace:



8. OPERACIONES DE ASIGNACIÓN

Conoces los tipos de datos, las expresiones, etc. usadas en algoritmia, pero necesitas la forma de almacenar un dato en una variable o una estructura de datos, normalmente se suele usar el signo “=” con diferentes lenguajes de programación o una flecha, por ejemplo, con pseudocódigo, pero también se pueden usar otras formas de hacerlo dependiendo del tipo de estructura con la que trabajes.

Es una de las formas de proporcionar valores a una variable. Se representa con el operador: \leftarrow

Por ejemplo:

```
Var a, b, c: entero  
a  $\leftarrow$  (3+b)*c
```

La operación de asignación es una operación *destructiva*, ya que el valor que tuviese anteriormente la variable será modificado por el valor obtenido o asignado.



EJEMPLO PRÁCTICO

Se tienen 3 variables: a, b, c. Escribir las instrucciones necesarias para intercambiar entre sí su valor del siguiente modo:

- b debe tomar el valor de a
- c debe tomar el valor de b
- a debe tomar el valor de c

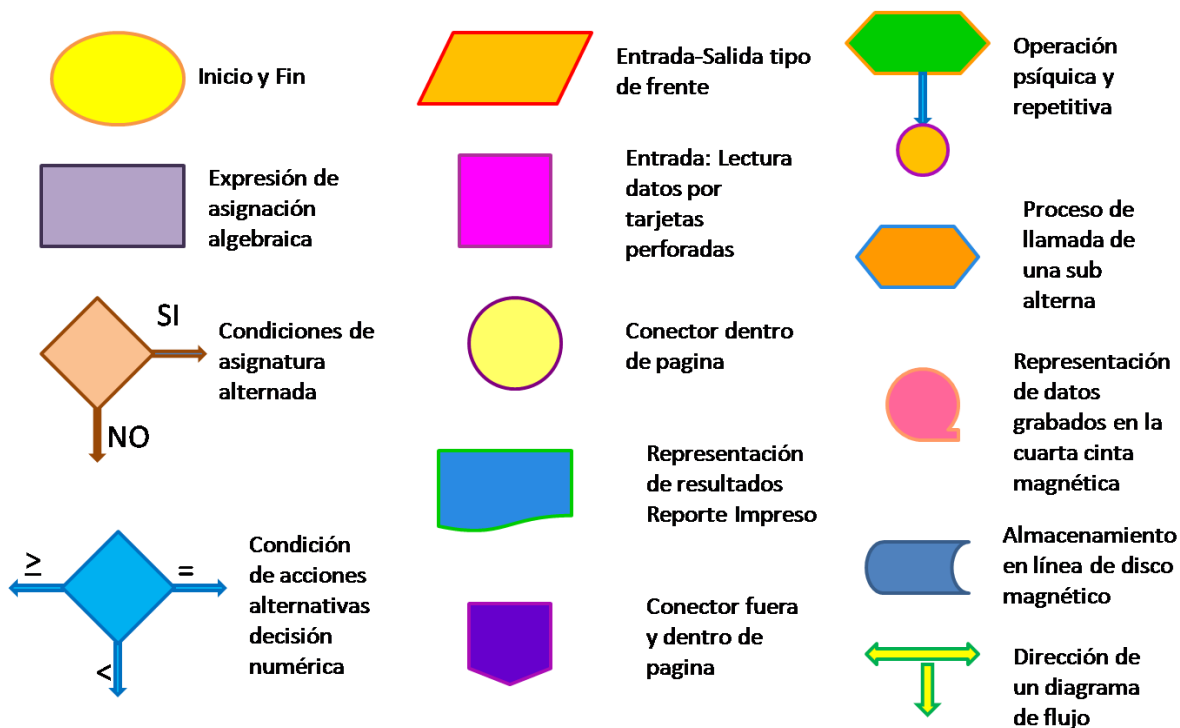
```
Var a, b, c, aux: entero  
aux ← b  
b ← a  
a ← c  
c ← aux
```

9. REPRESENTACIÓN DE ALGORITMOS. SOFTWARE DE UTILIDAD

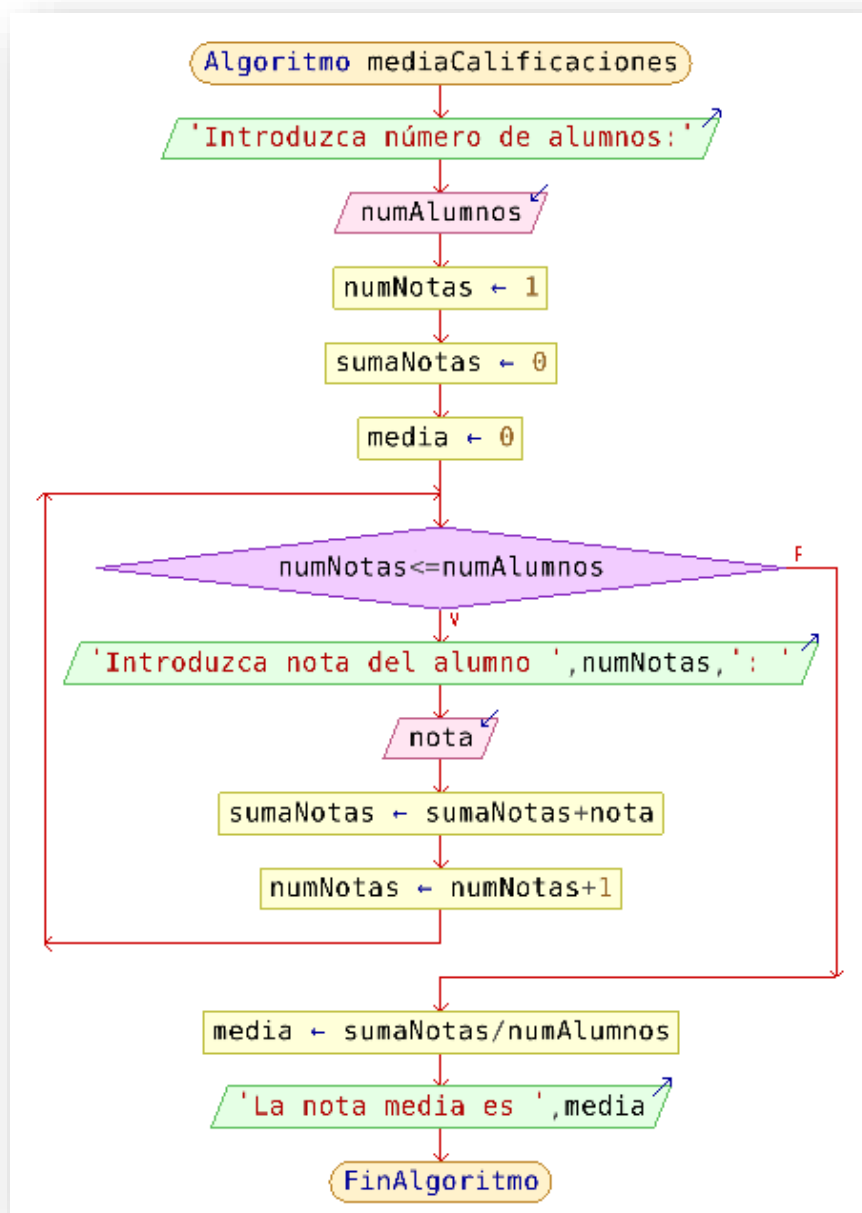
Ya te sientes cada vez más preparado para comenzar a programar la aplicación, pero como te sugirió tu amigo, debes empezar poco a poco, primero con la realización de algoritmos que usan un lenguaje más cercano al natural. Antes de nada, tienes que familiarizarte con los tipos de representación de algoritmos y los símbolos que se pueden utilizar y lo que significan. Además, hoy día hay aplicaciones software que ayudan a realizarlos, por tanto, no tienes que utilizar lápiz y papel e incluso te permiten ejecutar pseudocódigo y así poder comprobar resultados por pantalla, esto es una ventaja.

Hay varias formas de representar las instrucciones y el orden en un algoritmo:

1. **Diagramas de flujo:** es una de las técnicas de representación de algoritmos más antigua. Se basa en la utilización de símbolos gráficos denominados **cajas**, en las que escribimos las acciones que tiene que realizar el algoritmo. Estas cajas están conectadas mediante **líneas de flujo** que indican el orden de ejecución de las acciones:



Elementos de un diagrama de flujo.



Ejemplo de diagrama de flujo.



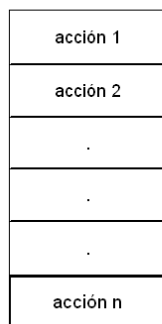
ENLACE DE INTERÉS

Puede ampliar información sobre los Diagramas de Flujo, podrá consultar la siguiente web:

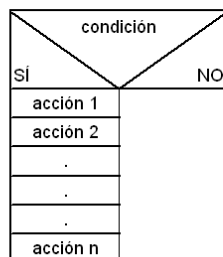


2. **Diagramas de N-S (Nassi-Shneiderman):** es idéntico al diagrama de flujo, eliminando las líneas de flujo, con las **cajas contiguas** unas a otras.

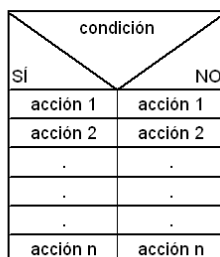
SECUENCIAL



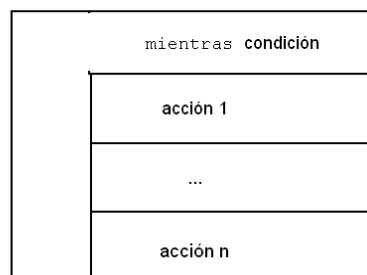
ALTERNATIVA SIMPLE



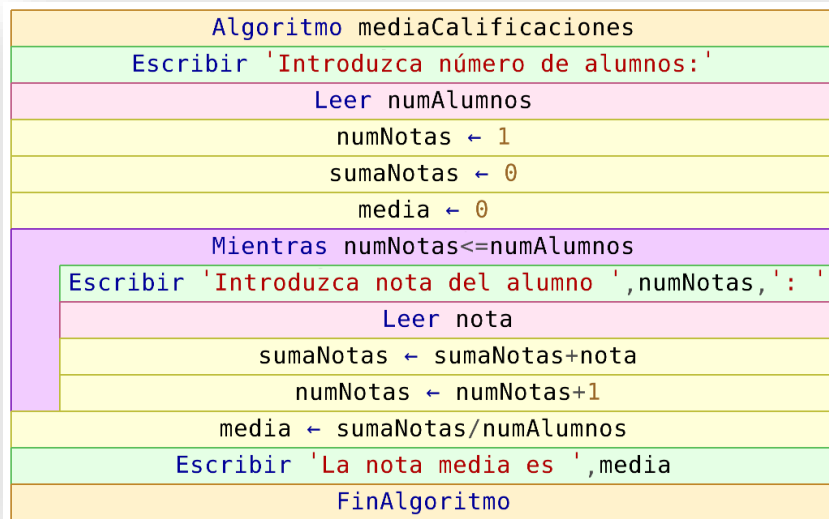
ALTERNATIVA DOBLE



ITERATIVA



Elementos diagramas N-S.



Ejemplo diagrama N-S que calcula la nota media de un número de alumnos.



ENLACE DE INTERÉS

Amplia información sobre estos diagramas en esta web:



PARA SABER MÁS

Conoce más sobre los diferentes diagramas de flujo que existen:



3. **Pseudocódigo:** este tipo de representación es el más utilizado, debido a que es el que más simplifica el paso de codificación al lenguaje de programación correspondiente.



EJEMPLO PRÁCTICO

En pseudocódigo se pueden usar los operadores aritméticos de la siguiente forma:

```
Variables:
op1, op2, result1: real
op3, op4, result2: entero

Inicio:
op1 <- 20.0
op2 <- 4.0
op3 <- 20
op4 <- 6

result1 <- op1 + op2
Escribir("La suma de op1 y op2 es = ", result1)

result2 <- op3 / op4
Escribir("La división de op3 entre op4 =", result2)

result2 <- op3 MOD op4
Escribir("El resto de dividir op3 entre op4 =", result2)
Fin
```



EJEMPLO PRÁCTICO

En pseudocódigo se pueden usar los operadores lógicos y relacionales de la siguiente forma:

```
Variables:
    op1, op2, result: entero

Inicio:
    op1 <- 20
    op2 <- 4

    Si (op1 / op2 > 4) Y (op2 > 0) Entonces
        Escribir("El resultado es mayor que 4")
    Sino
        Escribir("El resultado es menor que 4")
    Fin Si
Fin
```



EJEMPLO PRÁCTICO

Algoritmo que calcula la nota media de un número de alumnos introducido por teclado.

```
Algoritmo mediaCalificaciones
    Escribir 'Introduzca número de alumnos:'
    Leer numAlumnos
    numNotas<-1
    sumaNotas<-0
    media<-0
    Mientras numNotas<=numAlumnos Hacer
        Escribir 'Introduzca nota del alumno ',numNotas,': '
        Leer nota
        sumaNotas<-sumaNotas+nota
        numNotas<-numNotas+1
    Fin Mientras
    media<-sumaNotas/numAlumnos
    Escribir 'La nota media es ',media
FinAlgoritmo
```



EJEMPLO PRÁCTICO

Quieres diseñar un algoritmo que permita calcular el área y el perímetro de una circunferencia, sabiendo que el radio será un dato introducido por el usuario. Debes almacenar el valor pi cuyo valor es 3.1416 en una constante para que pueda ser accedida por el algoritmo. Igualmente necesitarás otras variables que almacenen el radio, el área y el perímetro. Tanto el área como el perímetro son datos de salida producidos por la realización de cálculos con datos de entrada como son el número pi y el valor de radio introducido por el usuario. Finalmente se desea que aparezca en pantalla el área y el perímetro resultante según el radio introducido por el usuario. Piensa y resuelve.

```
Algoritmo area_y_perimetro
    numeropi<-3.1416
    escribir ("Introduzca la medida del radio")
    leer radio
    area <- numeropi*radio^2
    perimetro <- 2*numeropi*radio
    escribir 'El area de la circunferencia es ', area, ' y el
    perimetro es ', perimetro
FinAlgoritmo
```



ENLACE DE INTERÉS

Puedes ampliar información sobre la representación de algoritmos, uso de variables y ejercicios para practicar leyendo este documento:



Actualmente existe una gran cantidad de software para la elaboración de **Diagramas de flujo**. Se recomienda el uso de los siguientes programas para la elaboración de Diagramas de Flujo:

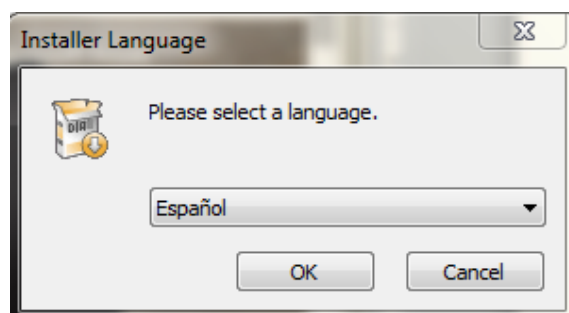
- **Software Comercial:** *Microsoft Office* ofrece 3 herramientas útiles para la elaboración de diagramas:
 - **Word:** Permite crear diagramas de flujo básicos a través de la opción de “Formas” que tiene un apartado especial para diagramas de flujo.
 - **PowerPoint:** Ofrece las mismas posibilidades de creación de diagramas.
 - **Visio:** Herramienta más sofisticada. Además de la simbología básica de diagramas de flujo, cuenta con una variedad de herramientas para elaborar otros tipos de diagramas.
- **Software No Comercial:** *DIA*. Solución rápida para la creación de diagramas de flujo además de otros tipos de diagramas y PseInt que genera automáticamente los diagramas de flujo asociados al pseudocódigo que se escribe.

Instalación del Software No Comercial DIA.

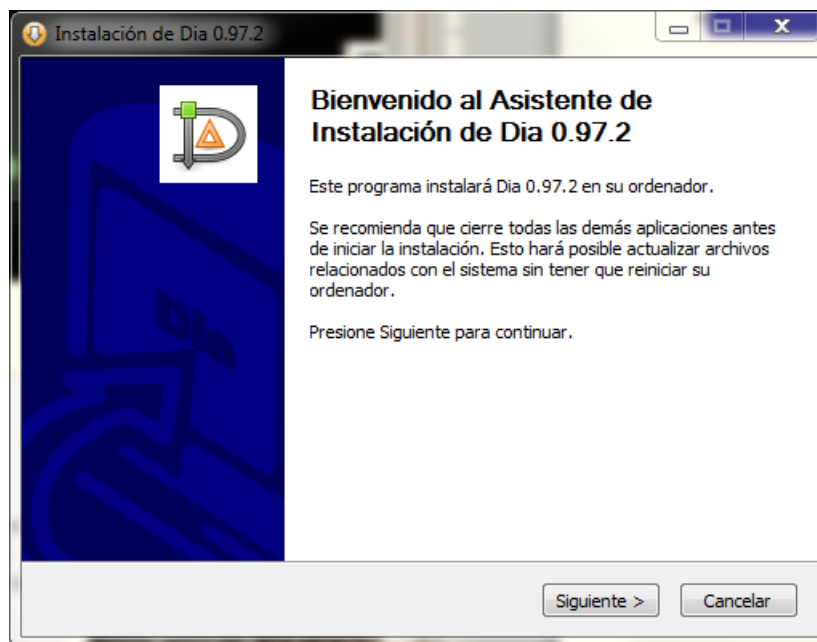
Se accede a la web oficial y se procede a realizar la descarga del instalador del software.



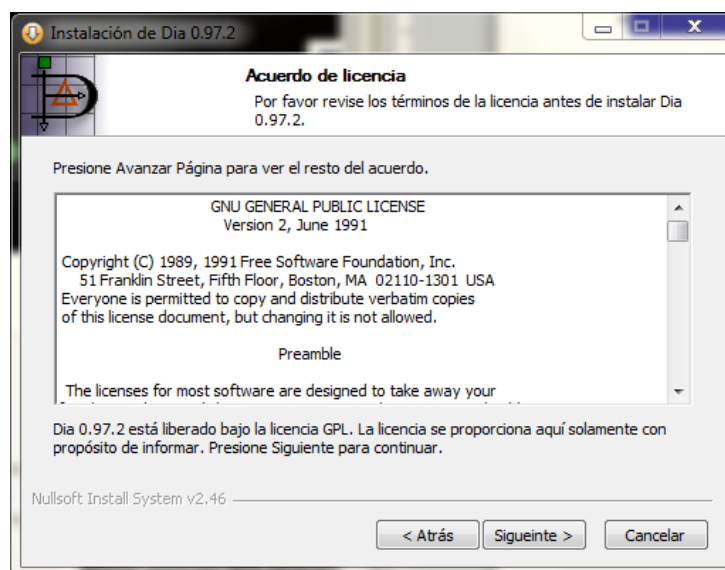
Se ejecuta la instalación del software, para ello se elige el idioma:



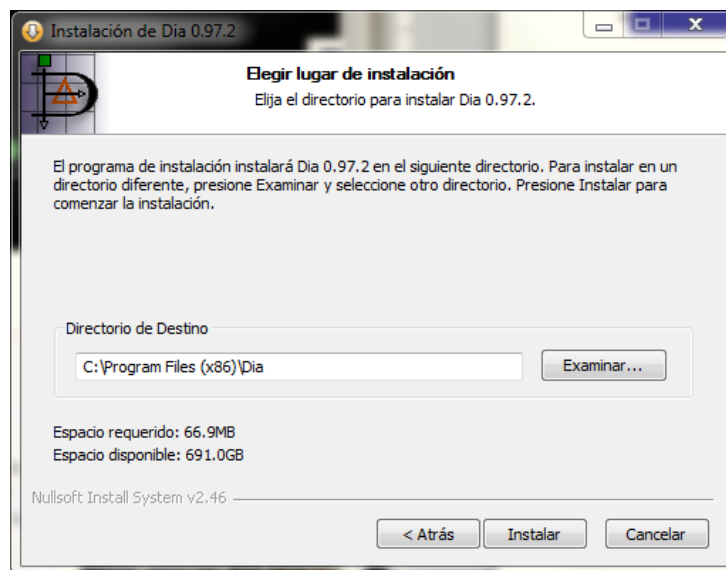
Se continúa con los pasos del asistente de instalación:



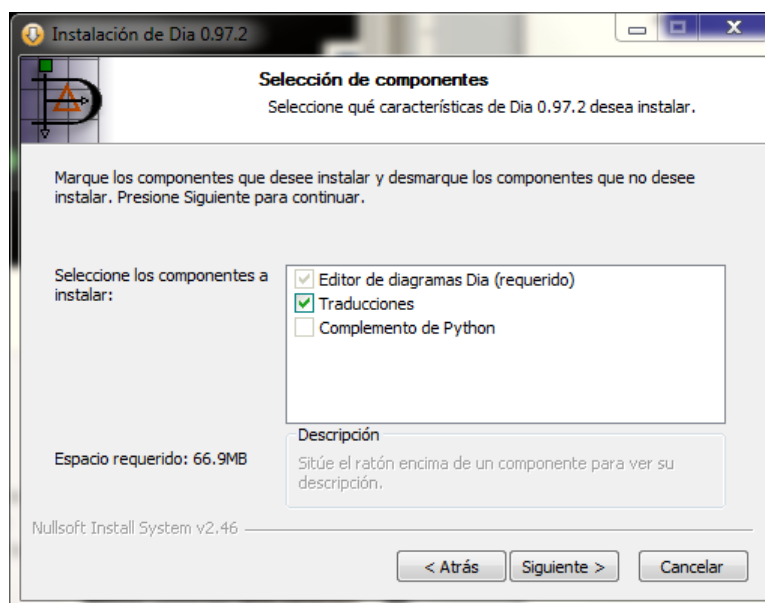
Se pulsa el botón Siguiente >

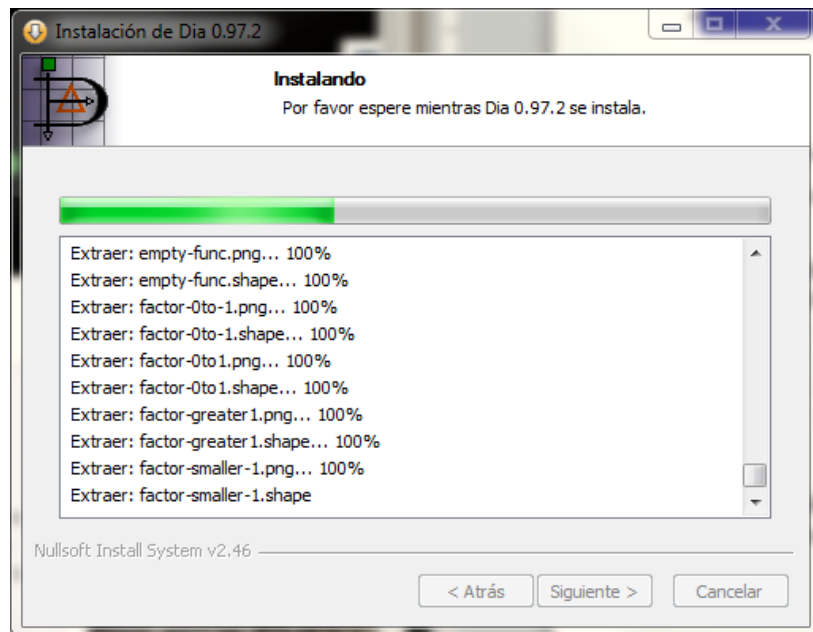


Se indica al instalador la ubicación en la que se desea instalar el software. Por defecto aparecerá la ruta `c:\Program Files(x86)\Dia` , pudiendo modificar la ubicación del mismo:

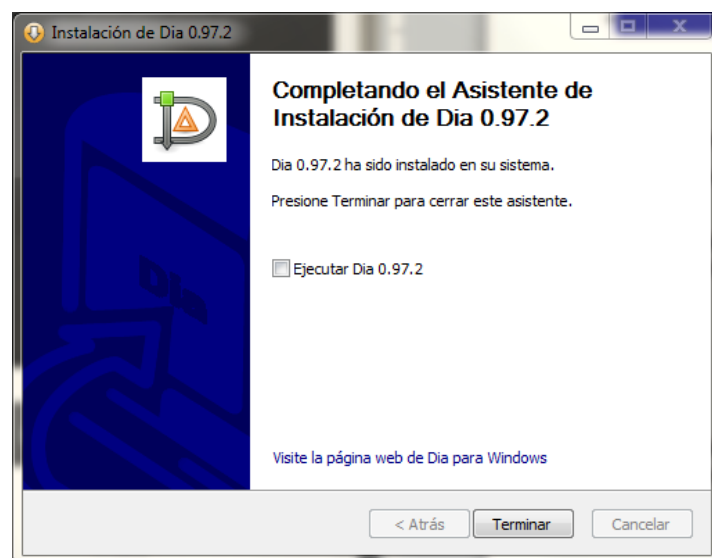


Se continúa con los pasos de la instalación, seleccionando las traducciones y pulsando el botón **Siguiente** >

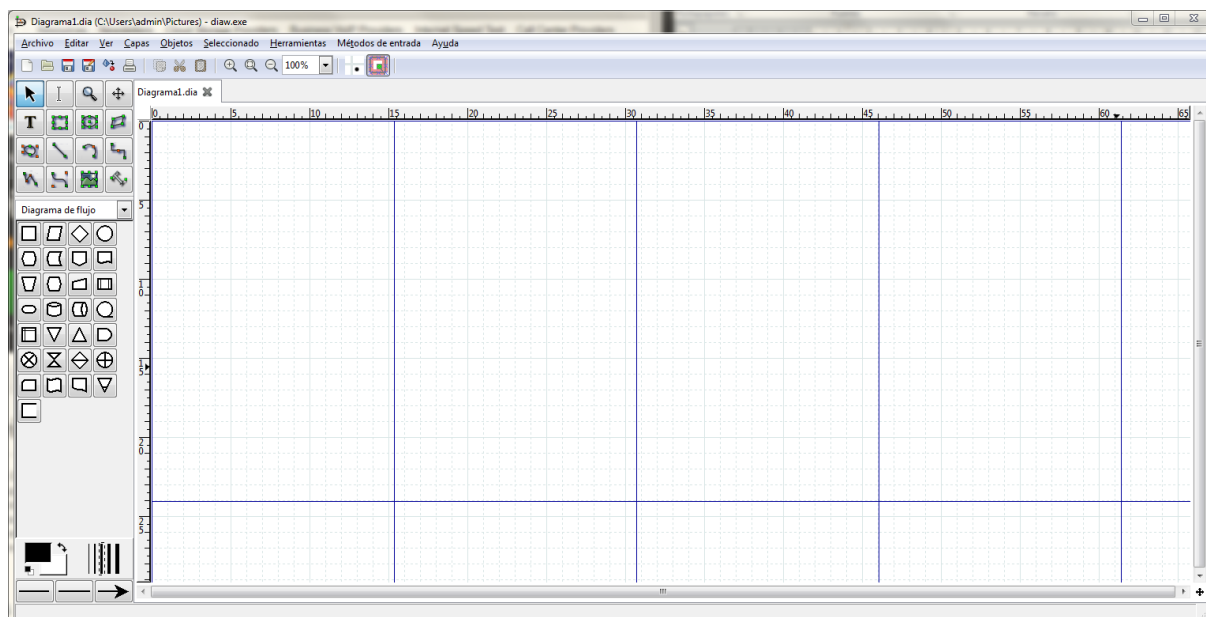




Una vez finalizada la instalación, ya se puede, por fin, utilizar el software de generación de Diagramas.



Al ejecutar el software, aparece la siguiente pantalla principal y se puede comenzar a generar los diagramas.



ENLACE DE INTERÉS

En esta dirección encontrarás un buen manual de usuario para el software DIA:



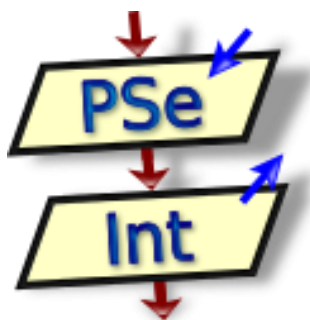


VÍDEO DE INTERÉS

En este vídeo encontrarás información sobre cómo realizar diagramas de clases UML con otra herramienta conocida llamada Miro.



A la hora de trabajar con **Pseudocódigo**, también se dispone de herramientas gráficas que permiten el desarrollo de la materia. En este caso se dispone del software *PSeInt*, el cual es un software libre educativo multiplataforma dirigido a personas que se inician en la programación.



Dicho software está pensado para iniciarse en la construcción de programas o algoritmos computacionales. El pseudocódigo se suele utilizar como primer contacto para introducir conceptos básicos como el uso de estructuras de control, expresiones, variables, etc., sin tener que lidiar con las particularidades de la sintaxis de un lenguaje real.

Este software pretende facilitar la tarea de escribir algoritmos en pseudolenguaje presentando un conjunto de ayudas y asistencias, y brindando además algunas herramientas adicionales que ayudan a encontrar errores y comprender la lógica de los algoritmos.

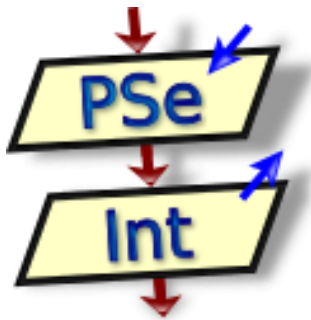


ENLACE DE INTERÉS

Para la instalación del software *PSeInt*, visita la web oficial:



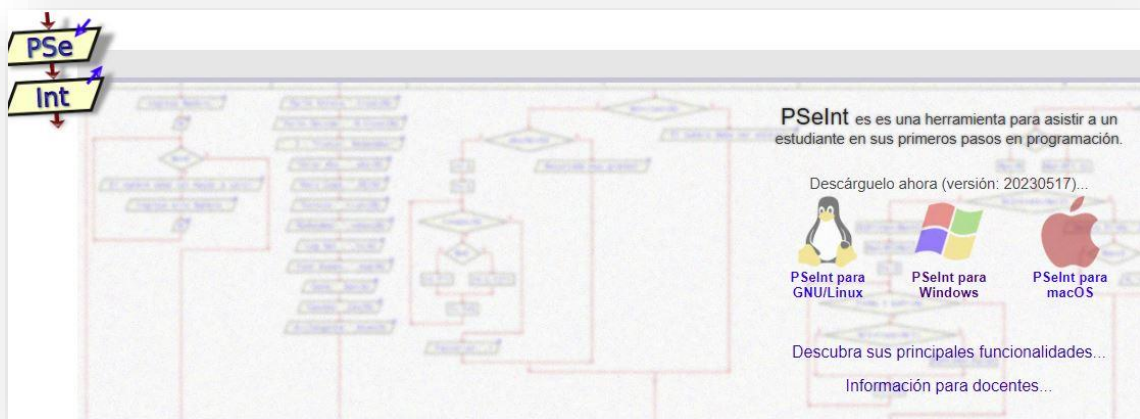
Instalación Software PSeInt



Se accederá a la web oficial del software PSeInt, a través del siguiente enlace:

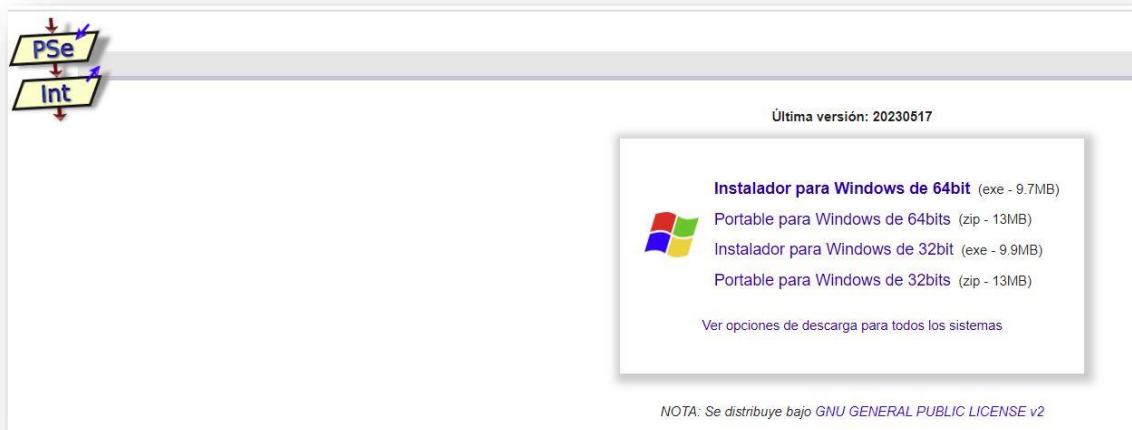
<http://pseint.sourceforge.net/>

y se realiza la descarga del instalador.



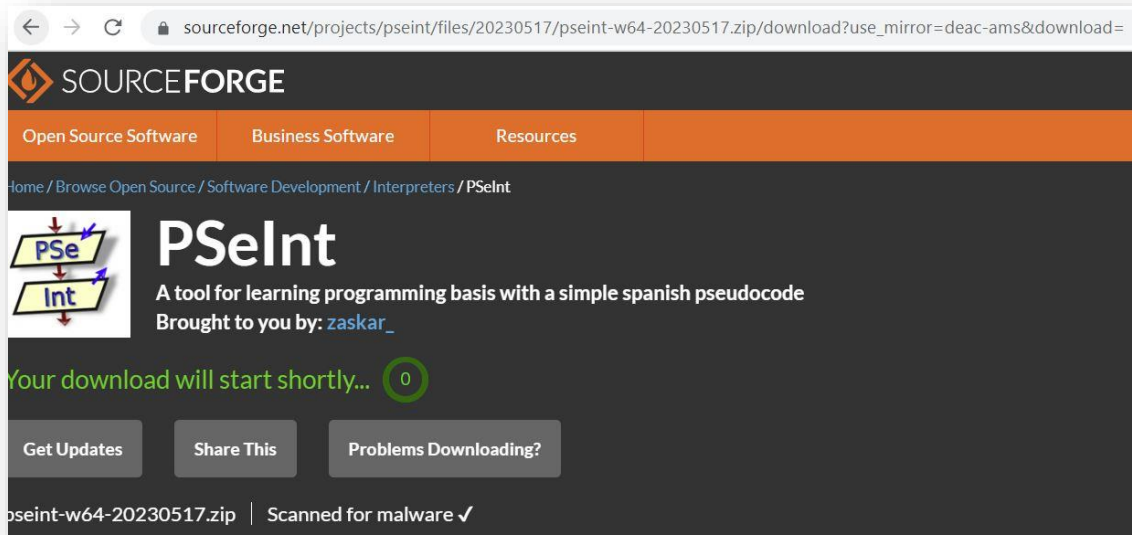
Instalación según Sistema operativo-PSeInt.
Fuente: Elaboración propia

Se selecciona el tipo de instalador necesario para el sistema operativo que se vaya a usar:



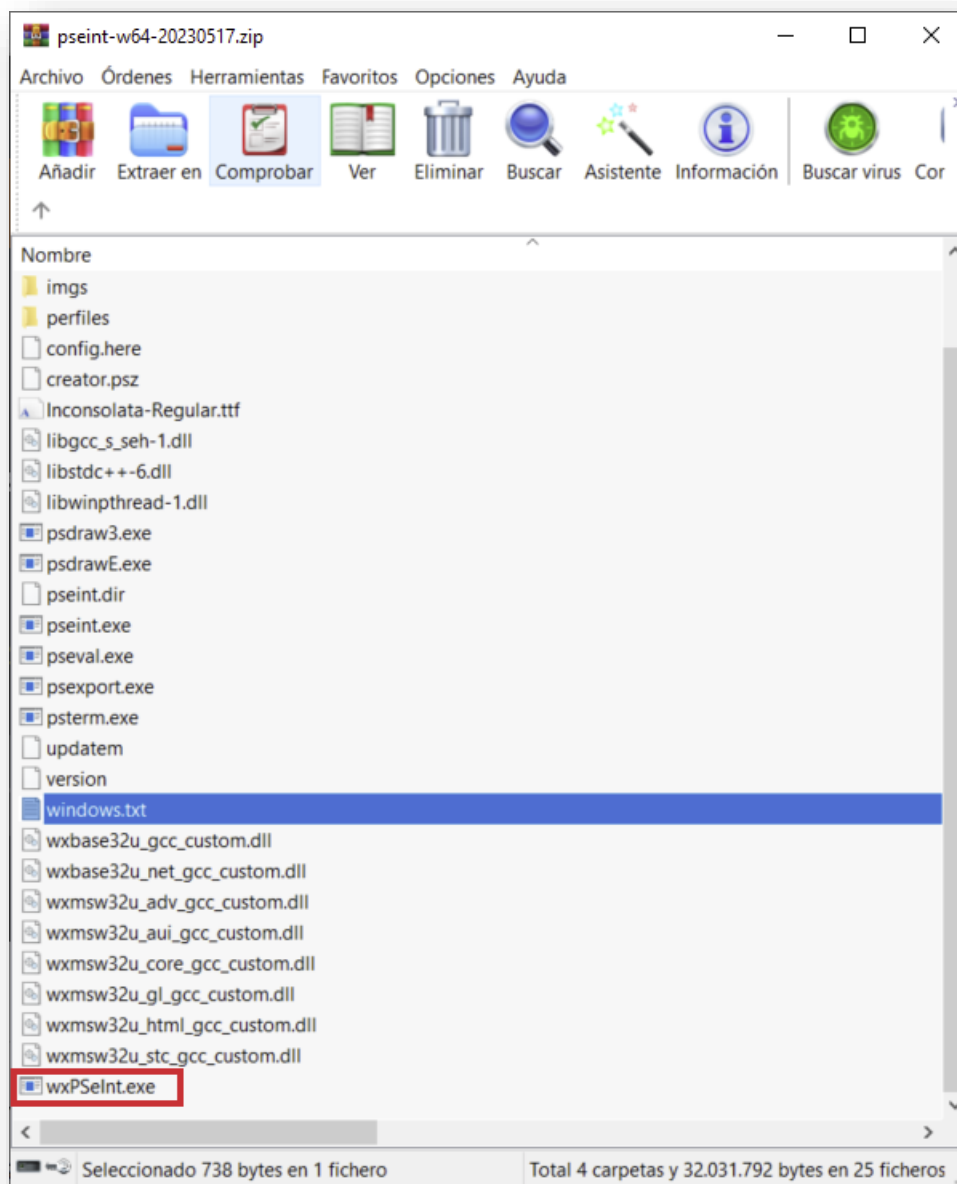
Instalación para Windows 64 bit-PSeInt.
Fuente: Elaboración propia

Para el sistema operativo Windows seleccionamos el instalador según sea nuestra versión de 32 ó 64 bits. Una vez seleccionado el instalador, se procede a la descarga a través de la web oficial.



Descarga software-PSeInt.
Fuente: Elaboración propia

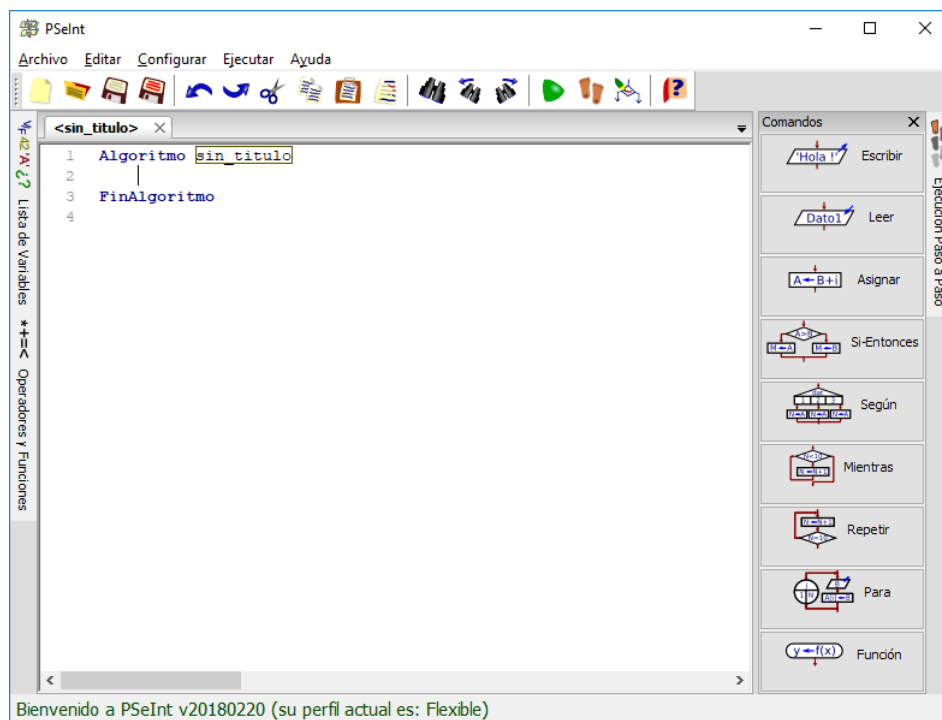
Se ha descargado un archivo .zip. Dentro de este archivo se encuentra el archivo wxPseInt.exe que hay que ejecutar.



Archivos de instalación-PSeInt.
Fuente: Elaboración propia

Se realiza la instalación ejecutando los pasos que indica el instalador y pulsando el botón Siguiente. Por defecto instalará el software en la carpeta indicada c:\Program Files(x86)\PSeInt\ , pudiendo el usuario modificar el destino o ubicación del software PSeInt

Una vez instalado en el sistema el software PSeInt, mostrará la siguiente pantalla principal:



Editor pseudocódigo-PSeInt
Fuente: Elaboración propia

Para empezar con PSEINT.

Las palabras **Algoritmo** y **FinAlgoritmo** son palabras reservadas y aparecen en un color diferente. Son palabras que tienen un significado para el programa y que se usan para determinar el inicio y el fin de nuestro algoritmo. Todas las instrucciones que haya entre estas dos palabras se ejecutarán de forma secuencial a menos que le indiquemos un cambio en el flujo mediante algún tipo de estructura cíclica o repetitiva.

Identificadores.

Debemos asignarle un nombre a nuestro algoritmo. Para ello utilizaremos la convención de nombres “**UpperCamelCase**”, que consiste en que la primera letra será mayúscula y si es palabra compuesta las primeras letras de cada palabra también (En Java los nombres de clases se forman de esta forma). Ej: AreaTrianguloRectangulo, CalcularTotalAPagarJuguetes, etc.

Sin embargo, para los identificadores de variables usaremos la notación “**lowerCamelCase**”, que consiste en lo mismo salvo por la primera inicial de comienzo de la primera palabra que será minúscula. Ej: precioJuguete, totalAPagar, numeroAleatorio, etc. Los nombres de identificadores pueden comenzar por; letras, números o guion bajo y NO deben contener espacios, operadores, palabras reservadas, acentos, eñes, diéresis o el nombre de otro identificador o función que se esté usando en otro lugar del código. Estos nombres deben ser explicativos en la medida de lo posible para ayudarte en revisiones posteriores.

¿Cómo definir el tipo de una variable?

Ejemplos:

- **Definir** acierto como **Lógico**.
- **Definir** suma como **Entero**.
- **Definir** nombre como **Cadena**.
- **Definir** total como **Real**.

Todas las variables que vayas a usar en tu programa debes declararlas. De esta forma estamos indicando el tipo de dato que va a almacenar la variable. Puedes definir varias variables de un mismo tipo en una misma línea.

Ej: Definir suma, num1, num2 como Entero.

También es aconsejable que las inicialices a 0 si son enteros, a falso si es lógica o booleana y a espacio en blanco (se ponen dos comillas juntas “ ”) si es de tipo texto.

Creando un nuevo algoritmo.

Abre el programa PseInt y pulsa la opción nueva del menú Archivo o en el dibujo ‘carpeta amarilla’ que se encuentra en la barra de herramientas.



Menú Archivo-PSeInt.
Fuente: Elaboración propia

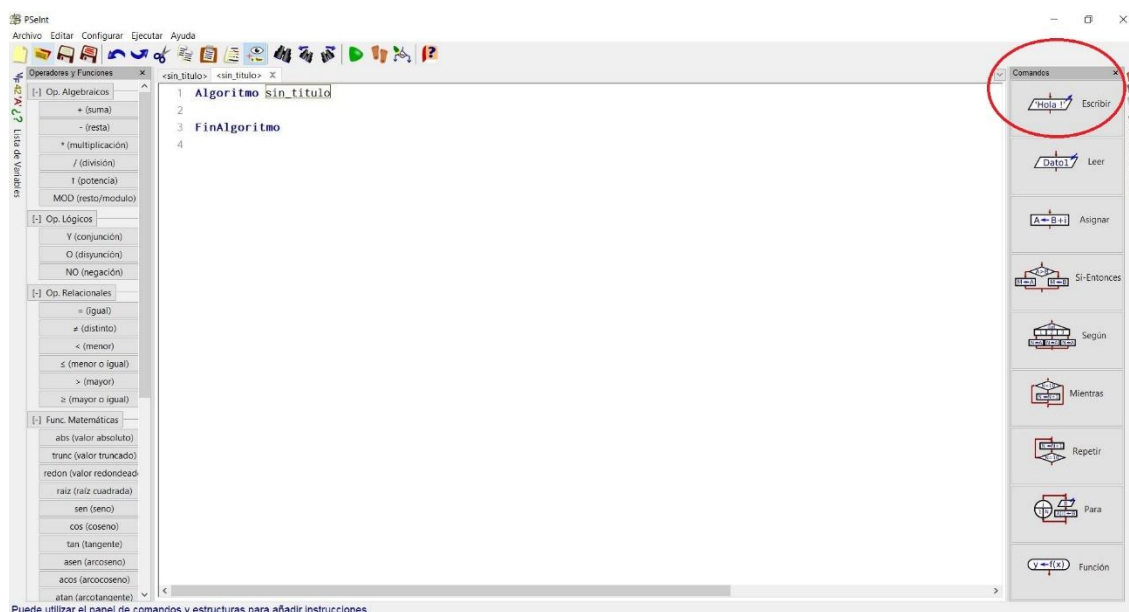
Para empezar con tu algoritmo debes tener en cuenta seguir unas reglas para que no haya errores de compilación ni de ejecución.

Puedes escribir directamente todo el código encerrándolo entre las palabras reservadas Algoritmo....FinAlgoritmo o hacer clic en el símbolo de diagrama de flujo que quieres usar(fíjate en la barra que se encuentra a la derecha de la pantalla) automáticamente aparecerá el texto correspondiente a la estructura seleccionada. Ahora solo tienes que escribir el pseudocódigo necesario dentro de dicha estructura.

Por ejemplo, si quieres usar la orden ...

Escribir “Introduce la base del triángulo”

Tienes dos opciones, bien empieza a escribir directamente o bien haz clic en el primer símbolo de la derecha (ver imagen) e inmediatamente aparece la orden 'Escribir' en pantalla en el lugar donde estaba el cursor. (Y así con el resto de instrucciones).



Editor pseudocódigo-PSeInt.

Fuente: Elaboración propia

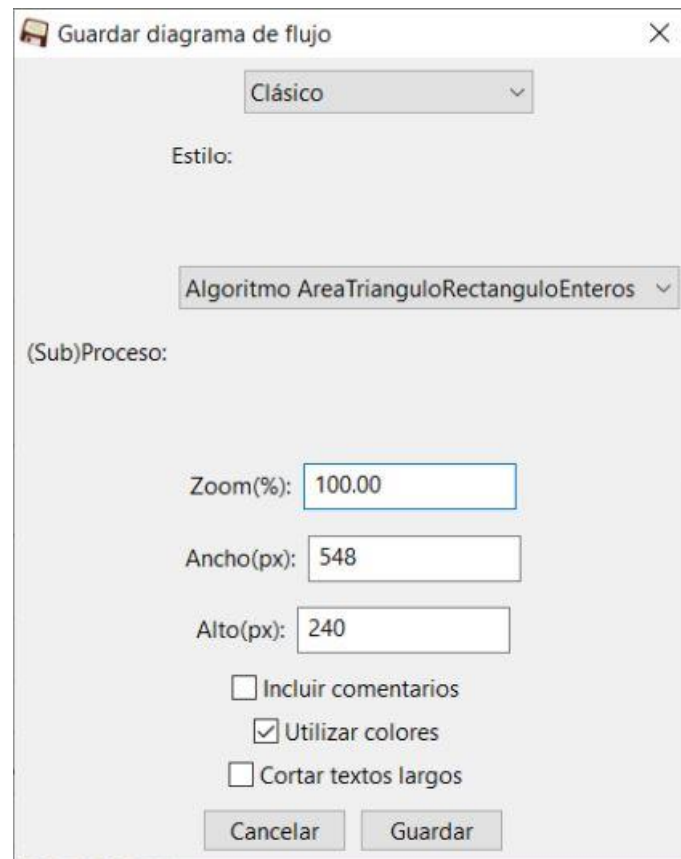
Con PseInt se puede crear de forma automática el diagrama de flujo, para ello pulsa el botón marcado con un círculo rojo que se muestra en la imagen de abajo y se abrirá la ventana en la cual verás el diagrama de flujo correspondiente al pseudocódigo que llevas escrito. Para salir de esta ventana pulsa la 'X' del menú ó la tecla ESC.



Algo importante es que de vez en cuando guardes tu trabajo, le des el mismo nombre que le has dado al Algoritmo para no confundirte y lo almacenes en una carpeta para evitar posibles pérdidas.

Si quieres abrir un algoritmo ya existente usarás la opción Abrir y buscarás el algoritmo que tendrás almacenado en la carpeta donde los estés guardando. Los archivos generados con PseInt tienen extensión .psc.

También puedes guardar los diagramas de flujo. Éstos se guardan como imágenes mediante la opción guardar que puedes escoger del menú o de la barra de herramientas.



Menú Guardar-PSeInt.

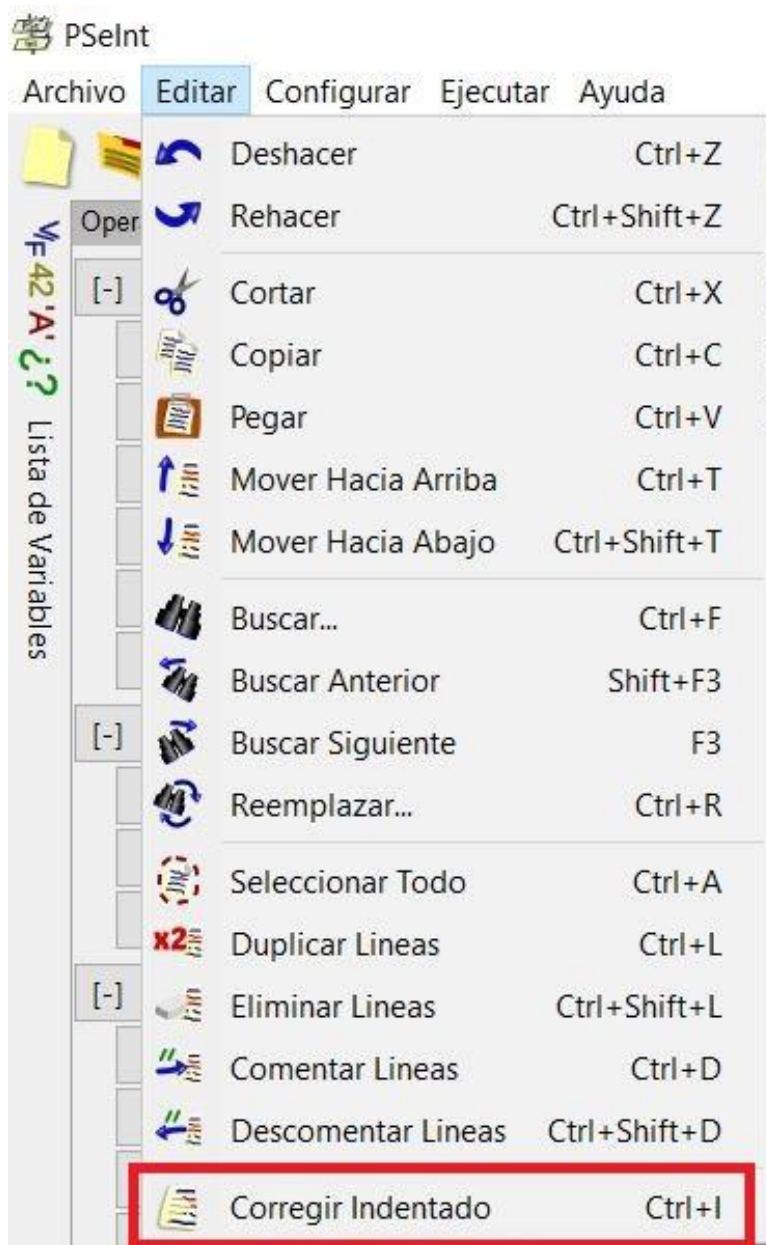
Fuente: Elaboración propia



NOTA DE INTERÉS

Prueba cada una de las opciones de la barra de menús y encuentra como cambiar el tipo de diagrama de flujo a otro diferente, como, por ejemplo, al tipo N-S visto en la unidad.

A la hora de programar es importante que el código se vea limpio y fácil de leer por lo que te recomiendo empezar desde ya a poner estas pautas en práctica. Para ello se hacen uso normalmente de tabulaciones. En caso de que el editor no lo haga automáticamente, seleccionaremos el código a **indentar** (proceso de añadir tabulaciones al código) y usaremos la opción '**Corregir indentado**', marcada con un cuadro rojo en la imagen de abajo.



Menú PSeInt.

Fuente: Elaboración propia



ENLACE DE INTERÉS

En esta dirección se encuentra la documentación oficial del software PSeInt:



ENLACE DE INTERÉS

Practica con los interesantes ejemplos que se incluyen en esta sección de la página oficial del software PSeInt:



VÍDEO DE INTERÉS

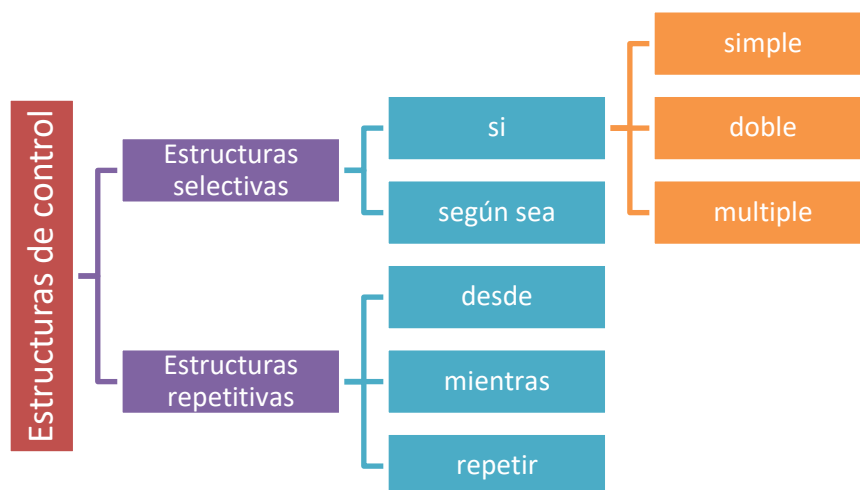
Visualiza cómo se utiliza la herramienta PSeInt y 100 ejercicios resueltos para inicializarnos en dicho software:



10. ESTRUCTURAS DE CONTROL

Con todo lo que has avanzado, ya puedes empezar a utilizar **PseInt** para escribir y ejecutar tus algoritmos, solo te hace falta conocer determinadas estructuras de control que controlarán el flujo de ejecución y tomarán decisiones en base a condiciones específicas. La filosofía de estas estructuras es muy parecida a las que utilizarás en Java. Sin perder tiempo comienzas a elaborar el pseudocódigo de las funcionalidades básicas utilizando PseInt, además puedes obtener los diagramas de flujo fácilmente que te vendrán muy bien cuando tengas estructuras anidadas y errores al compilar para ver la dirección del flujo de los datos.

Se pueden encontrar dos tipos principales de estructuras de control:



Las **estructuras selectivas** son las que permiten ejecutar una serie de instrucciones u otras dependiendo del valor de una expresión booleana o del valor de una variable numérica, de carácter o de texto.

Las **estructuras repetitivas** son las que permiten ejecutar una serie de instrucciones un número de veces indeterminado que va a depender de una expresión booleana, o bien un número predeterminado de veces.

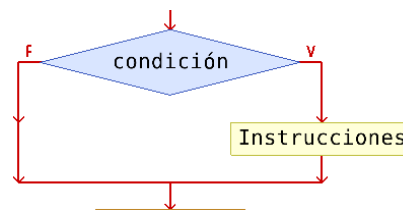
10.1 Estructuras selectivas o alternativas

Son aquellas que permiten la **ejecución de unas u otras acciones** dependiendo de que se cumpla o no una condición o dependiendo del valor que tome una determinada variable:

a) **Si:** permite ejercer una u otra acción dependiendo de que se cumpla o no una condición.

- **Simple:**

Si *condición* Entonces
 instrucciones;
Fin Si



La estructura Si...FinSi se denomina estructura de selección única porque ejecuta un bloque de sentencias solamente cuando se cumple la condición del Si.

- Si la condición es verdadera se ejecuta el bloque de sentencias.
- Si la condición es falsa, el flujo del programa continúa en la sentencia inmediatamente posterior al Si.

La condición es una expresión que evalúa un valor lógico, por lo que el resultado solo puede ser true (verdadero) o false (falso). La condición siempre se escribe entre paréntesis. La selección se produce sobre el bloque de sentencias delimitado por las instrucciones Si ...y FinSi.



EJEMPLO PRÁCTICO

Para practicar con la nueva estructura selectiva aprendida, se plantea el siguiente problema a resolver.

El usuario deberá introducir una edad por teclado y en caso de que esta sea mayor o igual a 18 se le indicará al usuario con un mensaje en pantalla que es mayor de edad.

Para dar solución al problema planteado necesitas definir(declarar) una variable, para almacenar la edad introducida por el usuario. La estructura de selección **si...finSi** permitirá tomar la decisión de imprimir el mensaje en pantalla que indique la mayoría de edad cuando el valor introducido por el usuario es mayor o igual a 18. Si la condición no se cumple(es falsa), es decir, la edad es un número que está por debajo de 18 el flujo de ejecución no entra en la estructura y la pasa por alto, ignorándola.

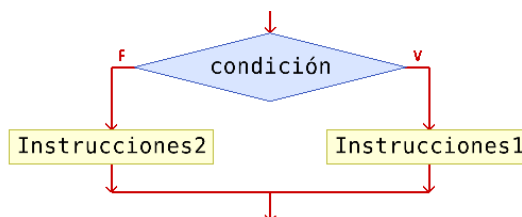
```
Algoritmo MayorDeEdad
  Definir edad como entero

  Escribir "Ingrese la edad:"
  Leer edad

  Si edad >= 18 Entonces
    Escribir "Es mayor de edad"
  FinSi
FinAlgoritmo
```

- **Doble:**

```
Si condición Entonces
  Instrucciones1;
Si no
  Instrucciones2;
Fin Si
```





EJEMPLO PRÁCTICO

Partiendo del ejemplo anterior, queremos que cuando la edad no sea mayor o igual a 18 se informe al usuario también, ya que anteriormente no se producía ningún aviso cuando la condición del `si...fin` era falsa y esto puede producir cierta confusión en el usuario que no sabe si es que el algoritmo no funciona. La estructura `si...sino...finSi`, nos ayudará a resolver este caso.

Si se cumple la condición (`true`), imprime por pantalla 'Es mayor de edad' y si no se cumple la condición (`false`), imprime el mensaje 'Es menor de edad'.

En ambos casos el usuario dispondrá de un mensaje apropiado según la edad introducida.

```
Algoritmo MayorDeEdad
  Definir edad como entero
  Definir mayorEdad como Logico

  Escribir "Ingrese la edad:"
  Leer edad

  Si edad >= 18 Entonces
    Escribir "Es mayor de edad"
  Sino
    Escribir "Es menor de edad"
  FinSi
FinAlgoritmo
```



EJEMPLO PRÁCTICO

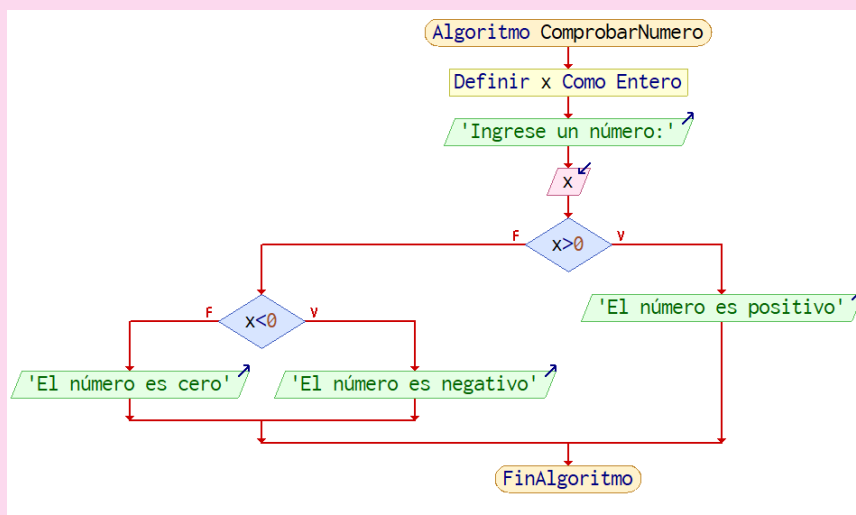
Con la estructura **si...sino...finSi**, solo tenemos dos opciones posibles para realizar acciones, cuando la condición es verdadera o cuando la condición es falsa, pero hay problemas a resolver que requieren validar otras opciones que se pueden dar cuando es verdadera o cuando es falsa, entonces usaremos esta estructura pero anidando unas dentro de otras.

El problema a resolver es el siguiente:

Un usuario introducirá un número entero por teclado y se mostrará en pantalla un mensaje que indique si el número es positivo, negativo o cero.

Se debe declarar una variable que almacene el número introducido por el usuario, a continuación, se lee el número y se almacena en dicha variable.

La validación comienza preguntando si el número es mayor que cero y si se cumple la condición, el usuario verá el mensaje 'El número es positivo' pero si la condición es falsa, entonces se tiene que volver a realizar una validación o pregunta porque si el número no es positivo con toda seguridad es porque será negativo o porque será un cero.



```

Algoritmo ComprobarNumero
  Definir x como entero
  Escribir "Ingrese un número:"
  Leer x
  Si x > 0 Entonces
    Escribir "El número es positivo"
  Sino Si x < 0 Entonces
    Escribir "El número es negativo"
  Sino
    Escribir "El número es cero"
  FinSi
FinSi
FinAlgoritmo
  
```

- **Múltiple:**

```
si condición1 Entonces
    Instrucciones1;
Sino si condición2 Entonces
    Instrucciones2;
    Sino si condición3 Entonces
        Instrucciones3;
    Sino
        Instrucciones4;
    Fin si
Fin si
Fin si
```



EJEMPLO PRÁCTICO

Investiga y practica las habilidades adquiridas con el uso de las estructuras `si..sino..finSi` anidadas, para realizar un algoritmo que muestre un texto que indique la categoría correspondiente a la nota introducida.

La nota introducida por el usuario puede ser un número con decimales por lo que la variable utilizada para almacenar este valor debe declararse como un dato de tipo real(en PseInt). De momento este algoritmo validará con categoría 'Excelente' cualquier nota mayor o igual a 9, es decir, que se podría introducir un 30, un 85,3 o cualquier otro número y siempre se produciría el mismo resultado('Categoría: Excelente').

Piensa en como podrías añadir otra validación más para que en caso de que el usuario Introduzca un número fuera del rango de 0 a 10 se muestre un mensaje que indique 'valor fuera de rango'.

```
Algoritmo PuntuacionEstudiante
    Definir calificacion como real

    Escribir "Ingrese la calificación del estudiante (0-10):"
    Leer calificacion

    Si calificacion >= 9 Entonces
        Escribir "Categoría: Excelente"
    Sino
        Si calificacion >= 8 Entonces
            Escribir "Categoría: Muy Bueno"
        Sino
            Si calificacion >= 7 Entonces
                Escribir "Categoría: Bueno"
            Sino
                Si calificacion >= 5 Entonces
                    Escribir "Categoría: Suficiente"
                Sino
                    Escribir "Categoría: Insuficiente"
                Fin Si
            Fin Si
        Fin Si
    Fin Si
FinAlgoritmo
```



RECUERDA

Cuando se utiliza una estructura de control como, por ejemplo, la estructura *si*, siempre tiene que terminar con un *FinSi*. En caso de emplear una estructura doble o compuesta se debe ir cerrando cada bloque *si...sino...* con su correspondiente *FinSi*.

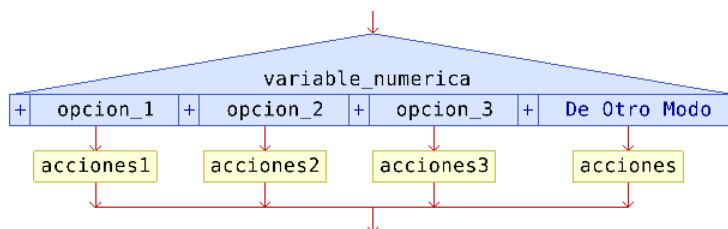


NOTA DE INTERÉS

Intenta realizar el diagrama de flujo del ejemplo práctico *PuntuaciónEstudiante* pero no utilices la utilidad de PseInt que lo genera automáticamente, de esta forma comprobarás si has asimilado la estructura condicional *si* anidada.

- b) **Según sea:** se usa cuando dependiendo del valor de una variable queramos ejecutar una u otras acciones.

Según *variable_numerica* Hacer
opcion_1: *secuencia_de_acciones_1*
opcion_2: *secuencia_de_acciones_2*
opcion_3: *secuencia_de_acciones_3*
 De Otro Modo: *secuencia_de_acciones*
 Fin Según





EJEMPLO PRÁCTICO

Te empiezas a plantear la posibilidad de crear un menú de opciones para que en función de la opción escogida se realicen una serie de acciones, para ello empleas la estructura `segun . . finSegun`. Estos tipos de menú son muy útiles porque presentan al usuario el conjunto de opciones que puede elegir y la acción esperada que se va a producir.

```
Algoritmo EjemploSegun
    Definir opcion como entero

    Escribir "Ingrese una opción (1-3):"
    Leer opcion

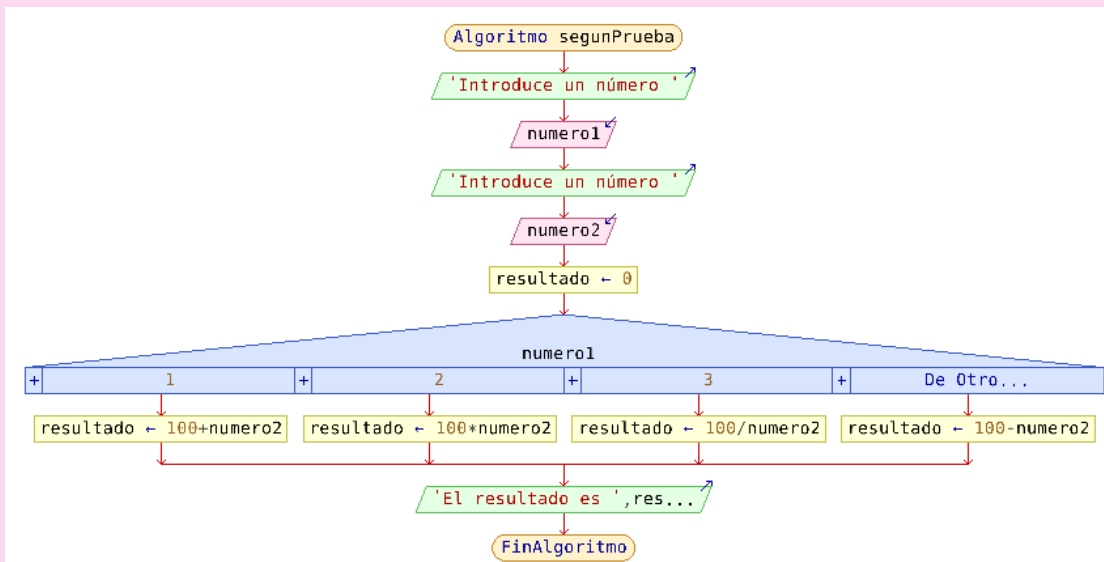
    Segun opcion Hacer
        1:
            Escribir "Seleccionaste la opción 1"
            // secuencia_de_acciones_1
        2:
            Escribir "Seleccionaste la opción 2"
            // secuencia_de_acciones_2
        3:
            Escribir "Seleccionaste la opción 3"
            // secuencia_de_acciones_3
    De Otro Modo:
        Escribir "Opción inválida"
        // secuencia_de_acciones
    Fin Segun
FinAlgoritmo
```




EJEMPLO PRÁCTICO

Te propones diseñar un algoritmo en pseudocódigo utilizando PseInt tal que, dados como datos dos variables de tipo entero, obtenga el resultado de la siguiente función: Aquí tienes el diagrama de flujo para guiarte

$$\text{Resultado} = \begin{cases} 100 + \text{numero2} & \text{si numero1} = 1 \\ 100 * \text{numero2} & \text{si numero1} = 2 \\ 100 / \text{numero2} & \text{si numero1} = 3 \\ 100 - \text{numero2} & \text{para cualquier otro valor} \end{cases}$$



RECUERDA

Las condiciones que controlan las estructuras selectivas devolverán siempre un valor booleano: verdadero o falso.



ARTÍCULO DE INTERÉS

Puede ampliar información sobre las estructuras de control selectivas y su sintaxis visitando la web:



10.2 Estructuras repetitivas

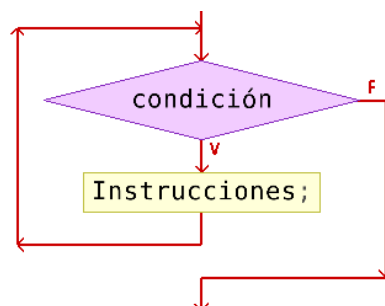
Estas estructuras permiten repetir un **número determinado** de instrucciones un **número finito** de veces.

- a) **Mientras:** se usa siempre que se quieran repetir una serie de instrucciones mientras se cumpla una determinada condición.

En caso de que la condición de entrada al bucle sea falsa(false) no se va a entrar en el mismo, por tanto, las instrucciones que se incluyen no se ejecutarán.

Esto quiere decir que el número de iteraciones de este bucle puede ir de 0 a n, es decir, puede ser 0 porque si la condición de entrada es falsa nunca se producirá la iteración. Y puede repetirse 'n' veces hasta que la condición sea falsa y haya una salida del bucle.

Mientras *condición* Hacer
Instrucciones;
Fin Mientras



Número de iteraciones: de 0 a n



EJEMPLO PRÁCTICO

A la hora de realizar una acción repetitiva puede ser conveniente preguntar al usuario cuando quiere parar, es por ello que se plantea el siguiente algoritmo utilizando la estructura Mientras...FinMientras.

Se pide que se introduzca un nombre y a continuación se salude a ese nombre, este proceso se debe repetir hasta que el usuario pulse una tecla 'n' ó una 'N'(mayúsculas o minúsculas), si pulsa cualquier otra tecla se da por hecho que quiere seguir repitiendo este proceso. Una vez el usuario ha decidido no continuar debe producirse la despedida mediante un mensaje en pantalla.

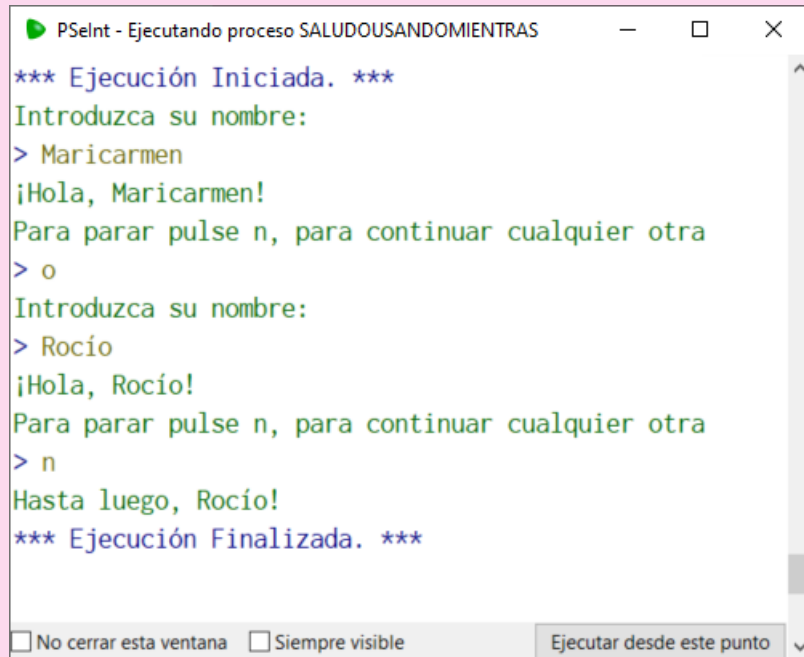
```
Algoritmo SaludoUsandoMientras
Definir nombre como Cadena
Definir respuesta como Cadena

respuesta <- ""

Mientras respuesta <> "n" Hacer
Escribir "Introduzca su nombre:"
Leer nombre

Escribir ";Hola, " + nombre + "!"
Escribir "Para parar pulse n, para continuar cualquier
otra"
Leer respuesta
respuesta <- Minusculas(respuesta)
FinMientras
Escribir "Hasta luego, " + nombre + "!"
FinAlgoritmo
```

A continuación, se puede ver un ejemplo de salida por pantalla de este algoritmo **SaludoUsandoMientras.psc** realizado con PseInt.



```
*** Ejecución Iniciada. ***
Introduzca su nombre:
> Maricarmen
¡Hola, Maricarmen!
Para parar pulse n, para continuar cualquier otra
> o
Introduzca su nombre:
> Rocío
¡Hola, Rocío!
Para parar pulse n, para continuar cualquier otra
> n
Hasta luego, Rocío!
*** Ejecución Finalizada. ***
```

☐ No cerrar esta ventana ☐ Siempre visible Ejecutar desde este punto



PARA SABER MÁS

PseInt incorpora unas funciones predefinidas que puedes ver en el menú “Operadores y funciones” que normalmente se encuentra en una de las barras laterales.

En el ejemplo anterior se ha usado una función que opera con texto llamada “minusculas(cadena)” que convierte la cadena indicada a minúsculas, es muy útil para que se validen tanto las mayúsculas como las minúsculas ya que se pasa todo a un formato común, en este caso a minúsculas.

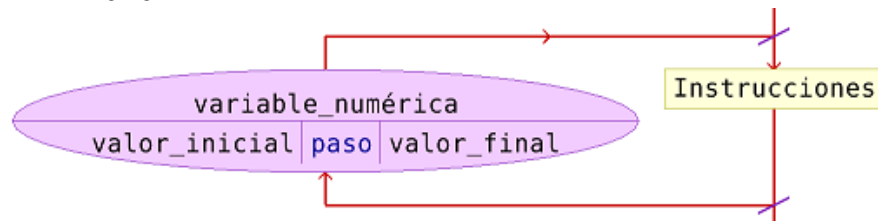
Haz clic en cada una de estas funciones de PseInt e investiga para que pueda servir cada una de ellas.

- b) **Para:** se usa cuando se conoce de antemano el número de veces que se quiere repetir algo.

Para *variable_numerica* <- *valor_inicial* Hasta *valor_final* Con Paso *paso* Hacer

bloque-de-sentencias

Fin Para



EJEMPLO PRÁCTICO

La estructura **Para...FinPara** es muy utilizada en programación cuando se conocen con certeza el número de iteraciones que debe realizar un bucle, por lo que tendrás que dominarla antes de empezar a emplearla con un lenguaje de programación. Practica y utiliza esta estructura para escribir por pantalla los números del 1 al 10.

```
Algoritmo EjemploFor
  Para i <- 1 Hasta 10 Con Paso 1 Hacer
    Escribir i
  FinPara
FinAlgoritmo
```

Piensa también en cómo se haría en orden decreciente, es decir, del 10 al 1.



EJEMPLO PRÁCTICO

A continuación, te proponemos que pidas al usuario un número positivo y que utilizando la estructura **Para...FinPara** se sumen los números comprendidos desde 1 hasta ese número y se muestre el resultado en pantalla. Este bucle se ejecutará 'n' veces. En este caso se conoce con certeza el número de iteraciones y va a coincidir con el número que introduce el usuario por teclado.

```
Algoritmo SumaHastaNConPara
  Definir num como Entero
  Definir sumaNum como Entero
  sumaNum <- 0

  Escribir "Introduzca un número entero positivo:"
  Leer num

  Para i <- 1 Hasta num Con Paso 1 Hacer
    sumaNum <- sumaNum + i
  FinPara

  Escribir "La suma de los números desde 1 hasta " +
  ConvertirATexto(num) + " es: " + ConvertirATexto(sumaNum)
FinAlgoritmo
```



PARA SABER MÁS

Dentro de la instrucción "Escribir" usada en la aplicación PseInt podemos usar una coma o un signo de suma para que aparezca al lado del texto mostrado el valor que contiene una variable o constante. Pero en el caso de usar el operador de suma(+) es necesario convertir el valor de dicha variable a un tipo texto.

Aquí tienes un ejemplo de cómo sería usando comas en vez de signos suma.

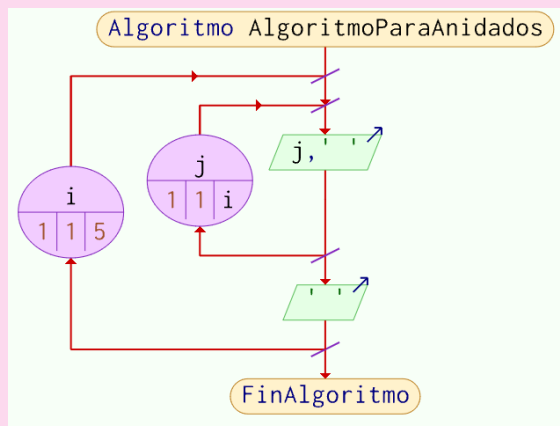
Ej: Escribir "La suma de los números desde 1 hasta ", num , " es: " , sumaNum



EJEMPLO PRÁCTICO

En próximas unidades veremos arrays unidimensionales y bidimensionales en Java y sería una buena recomendación comenzar a practicar con las estructuras para . . FinPara simples y anidadas. Este ejemplo te hará pensar un poco. El bucle exterior se ejecuta una vez y luego se entra al bucle interior. El bucle interior se ejecuta completamente antes de que el bucle exterior avance a su siguiente iteración. Este proceso continúa hasta que se cumpla la condición del bucle exterior.

Diagrama de flujo



```

Algoritmo EjemploParaAnidados
  Para i <- 1 Hasta 5 Con Paso 1 Hacer
    Para j <- 1 Hasta i Con Paso 1 Hacer
      Escribir j , " " sin Saltar
    Fin Para
    Escribir " "
  Fin Para
FinAlgoritmo
  
```

Al ejecutar este algoritmo se produce la siguiente salida por pantalla.

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
  
```



EJEMPLO PRÁCTICO

Seguimos entrenando el pensamiento lógico mediante la combinación de las estructuras vistas hasta ahora. Se quiere realizar la suma de los números de 1 hasta el número que indique el usuario, pero si el usuario introduce un número menor que cero o igual a cero se le comunicará que debe insertar de nuevo el número y se volverá a repetir el proceso. Cuando el número cumpla los requisitos (número positivo) se efectuará la suma y visualización del resultado.

Los comentarios están en tono gris y llevan delante dos barras inclinadas '//', que indican que son líneas no ejecutables y que solo informan al programador.

```
Algoritmo SumaHastaNConParaMasComplejo
    Definir num como Entero
    Definir sumaNum como Entero
    Definir bandera Como Logico

    sumaNum = 0
    bandera = Verdadero
    //El bucle se repetirá mientras el número introducido sea
    //negativo(bandera = Verdadero)
    //En el momento que se introduzca un entero positivo ya no se
    //repetirá el bucle mas
    Mientras bandera Hacer
        Escribir "Introduzca un número entero positivo :"
        Leer num

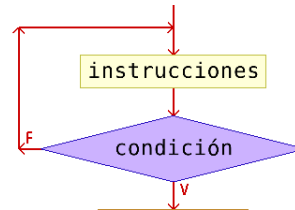
        Si num < 0 Entonces
            Escribir "El número es menor que cero.
                        Intente de nuevo."
        Sino
            Si num = 0 Entonces
                Escribir "El número es cero. Intente de nuevo."
            Sino
                //si el flujo de programa ha llegado aquí
                //es porque num es positivo
                //indica que ya se ha introducido un positivo
                // y no se va a seguir iterando el bucle
                //mientras
                bandera = Falso
                Para i <- 1 Hasta num Con Paso 1 Hacer
                    sumaNum <- sumaNum + i
                FinPara
                //ConvertirATexto(num) es una función que
                //convierte un número a texto, de esta //forma no se produce error
                cuando
                //al concatenar texto con número, de esta
                //se concatena texto + texto
                Escribir "La suma de los números desde 1
```



```
hasta " + ConvertirATexto(num) + " es: " + ConvertirATexto(sumaNu  
    FinSi  
    FinSi  
FinMientras  
FinAlgoritmo
```

- c) **Repetir**: se usa siempre que se quiera repetir una serie de instrucciones hasta que se cumpla una determinada condición.

Repetir
 instrucciones
Hasta Que *condición*



El cuerpo del bucle como mínimo se repite 1 vez. Número de iteraciones de 1 a n.



EJEMPLO PRÁCTICO

El desarrollo de videojuegos aporta una experiencia práctica valiosa que puede ser aplicada en otros proyectos de programación y en el mundo laboral, así que para repasar la estructura **Repetir..Hasta que**, sería genial comenzar con este sencillo ejemplo.

El juego consiste en que el usuario adivine el número secreto. Inicialmente se establece como número secreto el número 42 y se pide al usuario que introduzca un número.

Se le irán dando pistas acerca de si es mayor o menor que el número que tiene que adivinar para que realice de nuevo el intento y una vez lo encuentre se le felicita y se vuelve a preguntar si quiere seguir jugando. Todo el proceso se repetirá una y otra vez hasta que el usuario introduzca una letra 'n' mayúscula o minúscula cuando se le haga la pregunta para continuar o finalizar, pero si es cualquier otra letra (diferente a 'n' o 'N') continuará el juego.

```
Algoritmo AdivinanumSecretoConRepetirHastaQue
    Definir numSecreto, num como Entero
    Definir respuesta Como Caracter
    numSecreto <- 42
    num <- 0
    respuesta <- ""

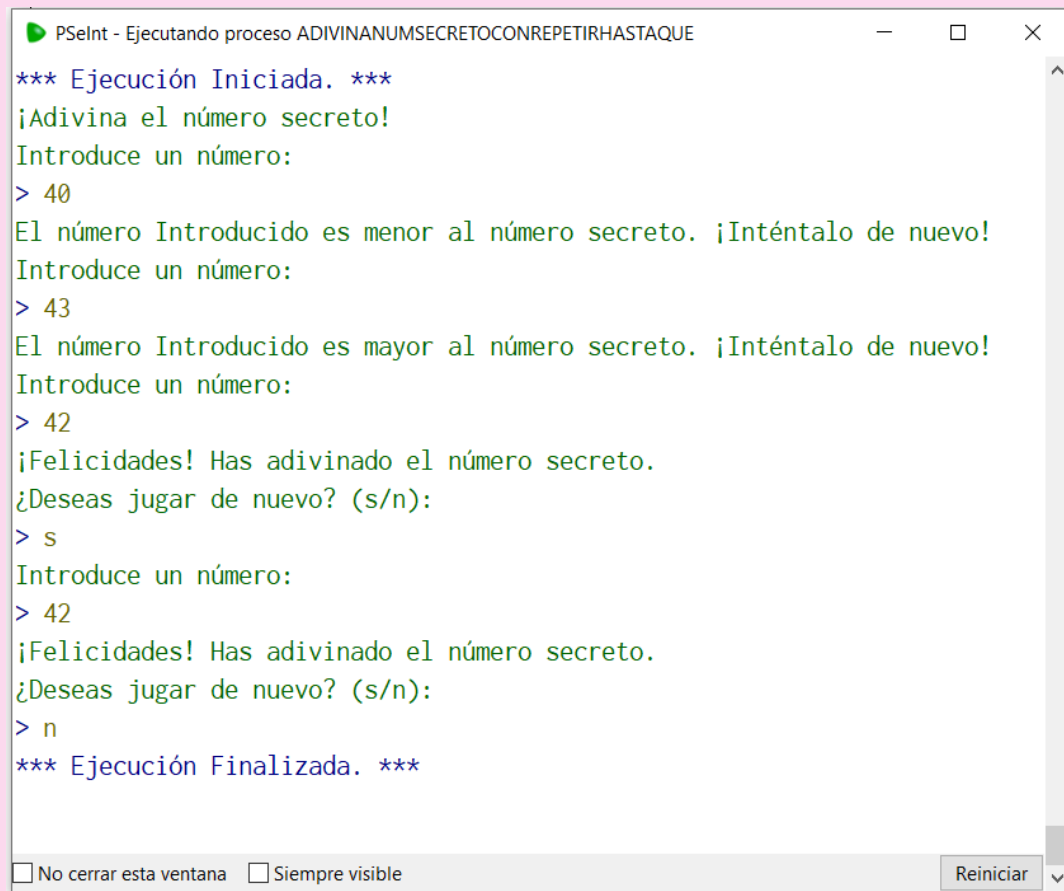
    Escribir ";Adivina el número secreto!"

    Repetir
        Escribir "Introduce un número:"
        Leer num

        Si num < numSecreto Entonces
            Escribir "El número Introducido es menor al número
secreto. ;Inténtalo de nuevo!"
        SiNo
            Si num > numSecreto Entonces
                Escribir "El número Introducido es mayor
al número secreto. ;Inténtalo de nuevo!"
            SiNo
                Escribir ";Felicidades! Has adivinado el
número secreto."
                Escribir "¿Deseas jugar de nuevo, pulsa n
para finalizar, cualquier otra
para continuar?:"
                Leer respuesta
                FinSi
            FinSi
        FinSi

    Hasta que (Minusculas(respuesta) = "n")
FinAlgoritmo
```

Ejemplo de salida por pantalla



```
PSeInt - Ejecutando proceso ADIVINANUMSECRETOCONREPETIRHASTAQUE
*** Ejecución Iniciada. ***
¡Adivina el número secreto!
Introduce un número:
> 40
El número Introducido es menor al número secreto. ¡Inténtalo de nuevo!
Introduce un número:
> 43
El número Introducido es mayor al número secreto. ¡Inténtalo de nuevo!
Introduce un número:
> 42
¡Felicidades! Has adivinado el número secreto.
¿Deseas jugar de nuevo? (s/n):
> s
Introduce un número:
> 42
¡Felicidades! Has adivinado el número secreto.
¿Deseas jugar de nuevo? (s/n):
> n
*** Ejecución Finalizada. ***

☐ No cerrar esta ventana ☐ Siempre visible Reiniciar
```

La estructura **Mientras...Hacer** y **Repetir...hasta que**, son muy parecidas, en la siguiente tabla se muestran algunas de las diferencias.

Estructuras de control	Diferencias	Número de iteraciones
Mientras <i>condicion</i> Hacer <i>instrucción1</i> ... FinMientras	El bucle se puede repetir ' 0 ' ó ' n ' veces, depende del valor de la condición en el momento de entrar al bucle	0 ó n veces
Repetir <i>instrucción1</i> .. Hasta que <i>condición</i>	El bucle se va a repetir 1 ó ' n ' veces. Cuando el flujo de control llegue aquí se va a permitir la entrada al bucle siempre, porque la condición de evaluación para salir o permanecer está al final del mismo.	1 ó n veces



NOTA DE INTERÉS

Es muy importante a la hora de programar el orden y claridad del código. Sobre todo, cuando se trabaja con varios desarrolladores colaborando en el mismo equipo de programación.



ARTÍCULO DE INTERÉS

Puede ampliar información sobre las estructuras de control repetitivas y su sintaxis visitando la web:





EJEMPLO PRÁCTICO

Por último y para demostrar que dominas las estructuras de repetición. Debes plantearte realizar un algoritmo que muestra en pantalla los números del 1 al 50, realizado con diferentes estructuras de iteración, Mientras, Para y Repetir.

Algoritmo Bucles

```
x <- 1
Mientras x <= 50 Hacer
  Escribir x
  x <- x + 1
Fin Mientras
```

```
Para x <- 1 Hasta 50 Con Paso 1 Hacer
  Escribir x
Fin Para
```

```
x <- 1
```

```
Repetir
  Escribir x
  x <- x + 1
Hasta Que x > 50
```

FinAlgoritmo



RECUERDA

Algunas de las estructuras estudiadas son equivalentes entre sí y un mismo problema se puede resolver utilizando varias de ellas.



PARA SABER MÁS

Realiza un algoritmo en pseudocódigo que muestre N números primos. N será un número entero que se pide al usuario. Si el usuario introduce un número negativo se debe mostrar un mensaje de error y volver a permitir que el usuario introduzca un nuevo número, este proceso se repetirá hasta que se introduzca un número correcto (entero y positivo).

Por último, se deben mostrar por pantalla los N primeros números primos.

Intenta utilizar de forma conjunta y en la medida de lo posible las diferentes estructuras repetitivas y condicionales aprendidas en la unidad (**mientras**, **Repetir..hasta que**, **Para y Si**)

RESUMEN FINAL

En esta unidad hemos aprendido a analizar los datos de entrada que va a necesitar un algoritmo para que, mediante una serie de operaciones con dichos datos, produzca los resultados de salida esperados.

Un algoritmo nos permite dar solución a un problema de una forma más sencilla. Los algoritmos se pueden representar de varias formas, en esta unidad hemos tratado los **diagramas de flujo**, los **diagramas N-S** y **pseudocódigo** para realizar el paso previo al desarrollo de un programa en un lenguaje de programación, utilizando herramientas software útiles para este fin.

Para poder comenzar a desarrollar algoritmos, se han tratado los diferentes **tipos de datos** más comunes entre los lenguajes de programación utilizados hoy día, como por ejemplo boolean, byte, short, int, long, double, float o char, también el concepto de **variable** y **constante**, así como el modo de combinar dichas constantes y variables para realizar **operaciones aritméticas y lógicas**.

Otro aspecto a destacar ha sido la incorporación de diferentes **estructuras de control** que se pueden clasificar en estructuras **selectivas o alternativas** y en estructuras **repetitivas**. Las estructuras selectivas nos permiten tomar decisiones y ejecutar diferentes bloques de código dependiendo de una condición específica entre las que destacan las aprendidas **si..finSi** y **según..finSegun**. Asimismo, las estructuras repetitivas o también conocidas como bucle o ciclo logran que un bloque de código se repita múltiples veces en función de una o varias condiciones de entrada o salida del bucle, destacamos las estructuras **mientras..finMientras**, **Para...finPara** y **Repetir...hastaQue**.

Una de las herramientas software usadas para adquirir soltura en el diseño de algoritmos de forma eficiente, ha sido **PseInt** que es una herramienta que nos ha ayudado a comprender y tener una visión más real de cómo funciona un programa, pero sin llegar a usar instrucciones complejas, que podrán traducirse y adaptarse a cualquier lenguaje de programación en un futuro.