2º de Bachillerato

Tecnologías de la Información y Comunicación

Contenidos

Conceptos básicos de programación: Las herramientas básicas del programador

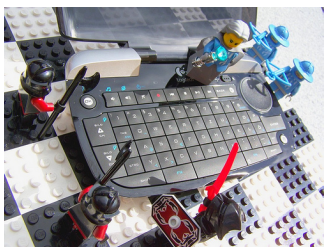


Imagen en Flickr de Sven Klaus con CC

En la actualidad **existe una gran multitud de lenguajes de programación**, creciendo su número además a un ritmo vertiginoso. Es muy complicado para un ser humano por tanto, conocer siquiera una mínima parte de los mismos. La tarea de actualización de sus propios conocimientos en un programador se antoja complicada y... eterna...

No obstante, **hay una serie de herramientas que son comunes a todos los lenguajes de programación** y su uso es esencial en todos ellos, por tanto, si se conocen desde un principio, el aprendizaje de cada lenguaje se hace bastante más fácil y llevadero.

Todo lenguaje es capaz de manipular, durante la ejecución de cada programa, ciertos **tipos de datos**, tanto en forma de valores fijos (los denominados "**constantes**") como variables (las ya conocidas y llamadas "**variables**"). Esos valores se utilizarán para realizar ciertas operaciones que lleven a conseguir una serie de resultados, por ello, el lenguaje necesitará un conjunto de **operadores** que lo posibiliten. Cada lenguaje usará mezclas de datos y operadores en forma de **expresiones** con una sintaxis concreta, a veces demasiado complejas para deducir lo que calculan, por lo que se suelen mezclar con **comentarios** explicativos que ayuden a los programadores a entender mejor el código fuente de los programas.

Estas herramientas se convierten pues en utensilios básicos para todo programador, es por ello por lo que te dedicarás en este tema a aprender las esencias de cada una de ellas, pues te abrirán las puertas de cada lenguaje de programación, permitiéndote abordarlo ya con ciertos conocimientos, lo que te agilizará el aprendizaje del mismo.

1. ¿Qué vas a aprender en este tema?

Cuando termines de estudiar este tema, estarás en perfectas condiciones de manejar las herramientas básicas de todo programador, independientemente del lenguaje que decidas estudiar en cada momento.

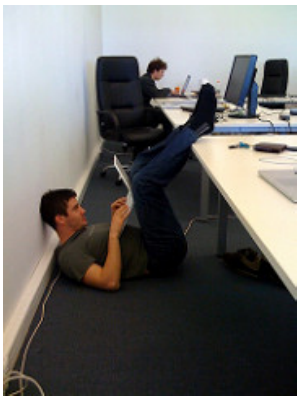


Imagen en Flickr de Miha Filej con CC

Podrás identificar qué tipos de datos necesitarán tus programas, cuántas variables tendrás que utilizar en ellos, estimar las constantes que vendrán bien para cada proceso, realizar expresiones fáciles y no tan fáciles, que te llevarán a conseguir que tus programas funcionen como tú deseas. Sabrás comentar adecuadamente las líneas de tus programas para hacer que éstos sean mucho más entendibles y comprensibles para los demás, y por supuesto para tí mismo.

En definitiva, al conocer estas herramientas, dejarás de ser un "bebé" en programación para sentar unas buenas bases, firmes y sólidas, de tu futuro aprendizaje en multitud de lenguajes de programación. Tu camino puede ser largo, pero ya tendrá bastantes menos escollos.

Reflexiona

Cuando termines de estudiar este tema, expresiones como las siguientes dejarán de ser un misterio para tí:

Imagen en Pixabay de Peggy_Marco bajo licencia CC0

"EJEMPLOS"+'*'+"VARIADOS"

2*(45-altura)/4-20

(12/temperatura+5/(coeficiente-2*(menor+10))

(40+precio)*21/100<=120*(coste+2)

10*altura/100>200 O ((base-10)*5+altura>=(coeficiente-3)/4-6

¿Podrías descifrarlas ahora?

Además, incertidumbres como las que te encontrarás a la hora de hacer un programa dejarán de serlo y podrás, entre otras cosas, elegir de forma adecuada los tipos de elementos a utilizar en ellos. ¿Podrías decir, por ejemplo, qué tipo de dato sería más conveniente para almacenar un teléfono, una edad o un DNI?.

Mostrar retroalimentación

Calma, no te preocupes, estas cuestiones y otras muchas, dentro de poco tiempo dejarán de preocuparte y de ser un misterio para tí. ¡Ánimo!

2. Datos y sus tipos

Según el diccionario de la RAE, **un dato** es cierta información sobre algo concreto que permite su conocimiento exacto. En informática esta definición se amplía, ya que un dato, además de dar información, lo hará dispuesta de tal manera que permitirá su tratamiento mediante un ordenador. Efectivamente, los ordenadores son máquinas que tratan datos pero ¿qué tipo de datos? En este apartado conocerás la diversidad de los mismos.

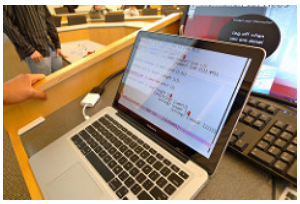


Imagen en Flickr de Jeff Dlouhy con CC

Un tipo de dato no es más que un dominio o rango de valores que admite una serie de operaciones, y al que el ordenador le da una forma interna de representación. Existen diversas clasificaciones de tipos de datos según diferentes criterios, entre ellas encontramos las siguientes:

- Según quién los define: tipos de datos estándar (vienen definidos en el propio lenguaje de programación) o tipos de datos definidos por el usuario (es el propio programador el que lo hace).
- Según su representación interna: tipos de datos simples o escalares (por ejemplo un número, o un carácter) o tipos de datos compuestos o estructrados (por ejemplo una fecha, o una palabra -compuesta por varios caracteres-).

En general, cada lenguaje de programación cuenta con una serie de tipos de datos estándares, tanto simples como compuestos. También pueden dar la oportunidad al propio programador de crear nuevos tipos de datos. De tal forma que las posibilidades son amplias. Por tanto, querer abarcar mucho en este momento no sería adecuado, pues te llevaría bastante tiempo. En lugar de eso, te centrarás en los tipos de datos más básicos y comunes, para ir profundizando más adelante. Así pues, saquemos factor común...

Para saber más

Se acaba de comentar que un tipo de dato es en realidad un dominio, es decir, un conjunto de valores. En la vida real, esos dominios suelen tener infinitos valores, piensa por ejemplo en los números, hay una cantidad infinita de ellos. ¿Cómo se plasma eso en un ordenador? Los ordenadores, por sus características físicas y de almacenamiento, no pueden representar un conjunto infinito de valores, por tanto se opta por "limitar" ese conjunto infinito de posibles valores. Así, cada tipo de dato en un ordenador tendrá una capacidad tope de almacenamiento, definida por el propio lenguaje de programación, lo que lleva a limitar el conjunto de valores que se permite representar. No te preocupes si esta idea no te queda totalmente clara, más adelante te toparás de nuevo con ella.

2.1 Tipos básicos de datos

Los tipos básicos de datos, también llamados primitivos, muy a groso modo, se pueden agrupar en 3: Los tipos de datos numéricos, los lógicos y los de texto (o alfanuméricos).

Dentro del grupo de los tipos de datos numéricos, a su vez, destacan dos tipos: los enteros y los reales.

Entre los tipos de datos de texto hay que mencionar dos: el tipo carácter y el tipo cadena (cadena de caracteres, o string).

Puedes ver de forma más esquemática esta agrupación de tipos de datos.

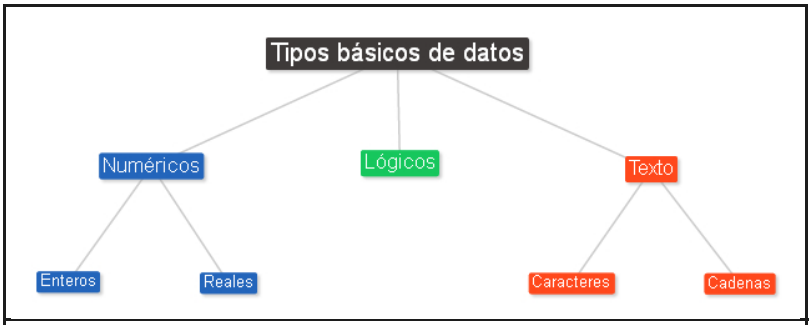


Imagen de creación propia bajo licencia Creative Commons

Este esquema en realidad es mucho más extenso, ya que cada lenguaje tiene multitud de tipos de datos, pero como se mencionó anteriormente, ahora debes captar las nociones elementales de los tipos de datos, por tanto, en principio, es preferible que te centres en unos pocos, los más generales y comunes:

- **Tipo de dato entero:** Son los números naturales positivos y negativos, más el cero. Se trata de un conjunto infinito de términos que en matemáticas usualmente se denomina "Z", compuesto por los números sin decimales. Cada término crece o decrece según para donde nos desplacemos en una unidad, por ejemplo 12, 13, 14, 15 o -3, -4, -5, -6. Son ejemplos 2, -4 y 0. En pseudocódigo se suele utilizar la palabra reservada "**entero**" para este tipo de datos.
- **Tipo de dato real:** Son los que pueden tomar como valores a los números racionales o irracionales. Este tipo de datos admite decimales. En matemáticas se denomina conjunto "R" y es también un conjunto infinito de términos. En este conjunto entre dos términos siempre existen un número infinito de términos. Para éstos, en pseudocódigo, se usa la palabra reservada "**real**".
- **Tipo de dato lógico:** Un dato lógico es aquel que sólo puede tomar valor verdadero o valor falso, es decir que algo se cumpla o no. Un ejemplo puede ser una puerta de paso, que puede estar abierta (asociamos por ejemplo verdadero) o cerrada (falso en este caso por oposición al convenio anterior). Para referenciar este tipo de datos en pseudocódigo se usa la palabra reservada "**lógico**".
- **Tipo de dato carácter:**El conjunto de valores que representa este tipo de datos es el formado por cualquier carácter que pueda representar el ordenador. Normalmente se representan entre comillas, ya sean dobles o simples (dependiendo del lenguaje). En pseudocódigo, para este tipo de datos, se utiliza la palabra reservada "**carácter**".
- **Tipo de datos cadena:** Una cadena es una secuencia de caracteres y se representa también normalmente entre comillas. Los espacios en blanco dentro del entrecomillado también son caracteres que forman parte de la cadena, por ejemplo "El gato" es una cadena de 7 caracteres. En pseudocódigo se usa la palabra reservada "**cadena**" para este tipo de datos.

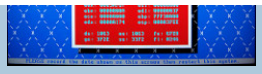


Imagen en Flickr de Rev.Xanatos con CC

subconjuntos de Z y R, fijándose por tanto máximos y mínimos para cada categoría de datos. Por ejemplo para los enteros suele ser el intervalo de los números comprendidos entre el -32768 y el 32767. El lenguaje de programación establecerá en cada caso dicho intervalo, es por ello por lo que se deberá consultar la documentación del propio lenguaje para conocer con exactitud los límites.

Cuando el ordenador maneja un tipo de datos numérico concreto, y se intenta asignar un valor que está fuera del subconjunto que representa, se produce lo que se conoce como un **error de desbordamiento (overflow en inglés)**. Cuando ocurre este error durante la ejecución de un programa, puede que el programa se detenga o puede que no. En caso de continuar, dará la sensación de no haber pasado nada, pero los resultados obtenidos no serían correctos, ya que el ordenador "intenta corregir" el desbordamiento asignando automáticamente un número que sí pertenezca al intervalo que se está representando en ese conjunto de datos (normalmente un cero).

Por ejemplo, supongamos que durante la ejecución de un programa, una operación que deba dar como resultado un número entero, se pasa del intervalo asignado a los números enteros (entre el -32768 y el 32767) y da un número mayor que el límite superior del intervalo (por ejemplo 45214), en ese caso se producirá un error de desbordamiento y la máquina asignará un cero a esa operación, y no el resultado real (45214). Lo sorprendente es que el ordenador no para la ejecución del programa, por ese motivo, esos errores son difíciles de detectar en nuestros programas.

Para saber más

Para complementar los contenidos y profundizar más sobre los mismos, puedes indagar sobre los siguientes aspectos:

- **Más sobre tipos de datos**: Existen diferentes tipo de datos como ya has visto. Para almacenar números reales (con decimales) has visto el tipo de dato "real" (float en muchos lenguajes), aunque hay otros que también pueden almacenar números con decimales y permiten ampliar el intervalo o subconjunto de los mismos que es capaz de manejar el ordenador. Se trata del tipo de dato "**doble**" ("double" en bastantes lenguajes). Hay otros tipos que podrías investigar: **enumerados, subrango, tupla**...
- **Sobre conversiones de tipos de datos**: En ocasiones, un programa necesita manipular variables de tipos de datos diferentes en una misma expresión. Por ejemplo, imagina que se desea sumar una variable de tipo entero (que almacenará un número entero) con una variable de tipo real (que almacenará un número real). Como los operadores están definidos para un tipo de dato concreto, la máquina se puede encontrar con una tesitura: ¿qué tipo de dato se debe tomar para cada dato? ¿y para el resultado final de la expresión?. Para ello se suele utilzar lo que se conoce como **conversiones de tipo**. Una conversión de tipo no es más que la transformación de una variable que está definida como un tipo de dato concreto, a otro tipo de dato, por ejemplo, de entero a real, o viceversa. Existen dos tipos de conversiones: **conversiones implícitas y conversiones explícitas**. Indaga sobre ellas.

2.2 Identificadores

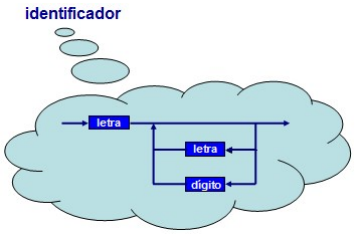


Imagen de creación propia bajo licencia CC

Los identificadores, en los lenguajes de programación, son palabras o textos que se utilizan para nombrar diferentes elementos del lenguaje. Cuando leemos el código fuente de un programa, escrito en un determinado lenguaje, incluso en pseudocódigo, encontramos multitud de términos y palabras que corresponden a identificadores. Los lenguajes de programación utilizan una serie de identificadores o palabras para asignárselas a diferentes elementos del propio lenguaje (como las instrucciones, los operadores, las constantes predefinidas,...). Son las llamadas **palabras reservadas del lenguaje** (en pseudocódigo por ejemplo, las palabras entero, real, lógico, carácter y cadena, son palabras reservadas del lenguaje). Los programadores utilizarán esas palabras reservadas para elaborar sus programas, pero además, manejarán otra serie de palabras definidas por ellos mismos (constantes, variables,...), teniendo ellos mismos la libertad de elegir qué palabra usar en cada caso, es decir, que ellos mismos son los que pueden elegir el identificador más adecuado en cada caso.

Los nombres de los identificadores deben seguir unas reglas de sintaxis, establecidas por los propios lenguajes de programación. Suelen ser éstas:

1. No pueden existir dos identificadores que se llamen igual. De aquí se puede deducir que las palabras reservadas del lenguaje no pueden ser usadas como identificadores creados por el programador.
2. Para nombrarlos podemos usar letras, dígitos numéricos e incluso algunos caracteres especiales (como subrayado bajo _), pero el primero no podrá ser un dígito numérico.
3. Dependiendo del lenguaje, se distinguirá o no entre mayúsculas y minúsculas. Es decir, que puede haber lenguajes en los que no haya distinción entre mayúsculas y minúsculas, y por tanto identificadores como "altura" y "Altura" sean en realidad el mismo, o en cambio, habrá lenguajes en los que sean dos identificadores distintos.

Importante

Aunque tú como programador, tendrás la libertad de elegir los identificadores a utilizar para tus variables y constantes, es muy conveniente que uses aquellos que tengan cierto significado, relacionados con el valor que van a almacenar. Eso hará que tu código fuente sea mucho más inteligible. Por ejemplo: si vas a necesitar una variable que almacene la altura de un rectángulo, no la llames de cualquier forma, lo más lógico sería llamarla "altura". Aunque este ejemplo te parezca tonto, no lo es, ya que a veces, por hacer las cosas a la ligera no nos paramos a elegir nombres adecuados, sino los más cortos o cómodos y al final obtenemos un código fuente poco o nada entendible.

Para saber más

suelen ser identificadores iguales o muy parecidos, lo que favorece el aprendizaje de múltiples lenguajes cuando se tienen claros los fundamentos de la tipología de datos:

PSEUDOCÓDIGO	PHP	C	JAVA	PYTHON
entero	integer	int	int	int
real	float	float	float	float
booleano	boolean	-	boolean	bool
carácter	-	char	char	chr
cadena	string	-	string	str

Imagen en Flickr de Marcos Gasparutti con CC

Como ya has visto en el tema anterior, una **variable** no es más que una zona de memoria que un programa utiliza para almacenar un valor que puede cambiar durante la ejecución. El programador, cuando realiza el programa será el encargado de decidir qué nombre o identificador se le asignará a esa zona de memoria. Al reservar esa zona de memoria hay que especificar qué tipo de dato será el que se almacenará en ella, y ese tipo no cambiará nunca. Por tanto, ese espacio se puede llenar con distintos valores a medida que el programa se ejecuta, aunque todos ellos serán del mismo tipo, que no es otro que el tipo de dato que el programador haya definido al crear la variable. Por tanto, cuando programes establecerás las variables que necesites y el tipo de dato que almacenará cada una.

Una **constante** en cambio es un valor que se almacena en una zona de la memoria pero que no varía durante la ejecución del programa. Un ejemplo podría ser el número PI, que siempre permanecerá con el mismo valor. Las constantes también llevan asociadas un nombre, es decir, un identificador. Cuando se crea una constante, se especifica el identificador o nombre de la constante y el valor que va a tener (y por tanto también el tipo de dato), luego, a medida que se vaya necesitando dicha constante en el programa, sólo hay que poner el nombre o identificador de esa constante.

A la acción de crear una variable (o una constante) por primera vez en un programa se le denomina **declaración de variable (o constante)**. La vida de las variables y constantes dentro de un programa abarcará desde que son declaradas por primera vez hasta la finalización del programa. Todo ese tiempo es lo que se llama **ámbito de la variable (o constante)**. El programador podrá hacer referencia a las variables y constantes utilizando sus identificadores, pero siempre dentro del ámbito de las mismas (o sea, desde que son declaradas hasta que el programa termina).

Importante

Un dilema que todo programador tendrá cuando está haciendo un programa será qué tipo de elemento, variable o constante, y de qué tipo de dato, se necesita en cada caso. La respuesta a la primera parte es simple: Si el elemento no va a variar a lo largo de la ejecución del programa, el elemento a utilizar será una constante, en caso contrario se deberá utilizar una variable.

La respuesta a la segunda parte del dilema (qué tipo de dato será el más adecuado), es algo menos sencillo, pero resolviendo una serie de cuestiones claves y siguiendo unas reglas básicas y elementales podrás decidirlo sin mucha dificultad. Aquí tienes un guión:

1. **¿El dato es de tipo texto o puede llevar algún texto dentro?** Si es así será de tipo **carácter o** de tipo **cadena**, dependiendo de si se va a almacenar un sólo carácter o varios.
2. **¿El dato a almacenar solo admite dos valores posibles?** Si ocurre esto, podemos elegir un tipo de dato **lógico o booleano**, que encaja perfectamente con la cantidad de datos posibles a almacenar (verdadero o falso).
3. **¿El dato que necesitamos almacenar es un número?** En ese caso está clara la decisión, aunque ahora viene una segunda cuestión, ¿qué tipo de número?:
    a. Si el dato no lleva decimales, escogeremos un tipo de dato **entero**.
    b. Si lleva decimales, sería un tipo de dato no entero, es decir, **real**, doble u otro tipo de dato que el lenguaje admita con números decimales.

Llegados a este punto ya sabrás el tipo de dato o lo tendrás casi claro, pero ahora viene **la clave para ser un buen programador**.

Debes observar el lenguaje elegido y ver los tipos de datos que ofrece el mismo, porque en ocasiones pueden servirte varios.

Piensa que necesitamos, por ejemplo, almacenar un dato numérico, en ese caso, los lenguajes tendrán disponibles varios tipos de datos enteros y varios tipos de datos numéricos con decimales, ¿cuál elegimos entonces?. Fácil, **elige el tipo de dato que te permita almacenar los valores de forma correcta, sin que se produzcan desbordamientos de datos (overflow) pero que representen a un intervalo de datos menor, ya que serán los que ocupen menos espacio en memoria**.

Esta regla parece inocente e innecesaria pero no lo es, piensa que el mejor programador es el que es capaz de hacer el mismo programa que los demás pero haciendo que éste necesite menos recursos de la máquina.

Curiosidad

Hay lenguajes de programación que obligan al programador a declarar todas las variables y constantes que utilizará en su programa. En cambio, existen otros lenguajes que no obligan al programador a declarar las variables y constantes, sino que el propio lenguaje "autodeclara" las mismas la primera vez que el programador las usa en su programa. En este último caso, a las variables y constantes se les suele asignar, dependiendo del lenguaje, o bien un tipo de dato acorde al valor almacenado esa primera vez, o un tipo de dato estandarizado y prefijado por el propio lenguaje.

En PSeInt existe la posibilidad de configurar el entorno para distintas versiones de pseudocódigo, que van desde la más flexible a otras más estrictas, pasando por las versiones que suelen usarse en distintas instituciones y universidades. **Puedes usar la versión flexible al principio**, que es bastante permisiva ya que puedes utilizar variables y constantes sin tenerlas que declarar previamente. **Luego**, si lo estimas oportuno, **podrás ir cambiando a otras versiones más estrictas**.

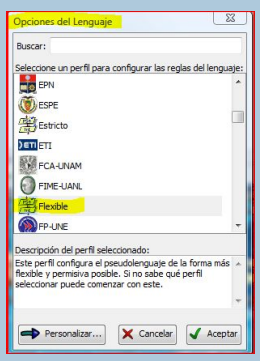


Imagen de creación propia

bajo licencia CC

Si lo prefieres también tienes la opción de crear un perfil totalmente personalizado para tu propio pseudcódigo. Tan sólo tienes que acceder a la configuración de las opciones del lenguaje y hacer clic en el botón de "Personalizar" el perfil que encontrarás en la ventana, bajo la lista de los distintos perfiles . Si te decides por personalizar tu propio perfil, encontrarás una lista de opciones a elegir, algunas de las cuales ya te sonarán bastante, otras las irás encontrando en los próximos temas, no te preocupes por ellas.

4. Operadores y Expresiones

Imagen en Flickr de Bob Simmons con CC

Como ya sabes, los tipos de datos tienen una serie de operaciones asociadas, de tal manera que con cada tipo de datos se puede operar de forma diferente, por eso es muy importante definir bien el tipo de dato que se almacenará en cada variable, pues el tipo de dato restringe el tipo de operación que se puede realizar con los mismos. Así, por ejemplo, las operaciones matemáticas son típicas de datos de tipo numérico. Cuando se realiza operaciones con los datos, se genera una expresión donde, además de los propios datos, aparecen también los signos correspondientes a las operaciones realizadas. A los datos con los que se opera se les llama **operandos**, a las operaciones se les denomina **operadores**. Aquí tienes algunos ejemplos:

12 + 14

43+3-17+5

7*14

(2+4)*3

((2-9)*11)*3

En el primer ejemplo aparecen dos operandos (el 12 y el 14) y un operador (el operador +). En el segundo existen cuatro operandos y tres operadores, etc. En algunas expresiones se introducen paréntesis. Éstos indican que las operaciones que figuran dentro son prioritarias y que han de realizarse antes. Así, el resultado del cuarto ejemplo será 18 resultado de 6*3, y no por ejemplo 14, que sería el resultado de la expresión si no hubiera en ella paréntesis, ya que el ordenador calcularía primero la multiplicación (4*3) y después la suma (2+12). A diferencia de matemáticas, aquí no se utilizan corchetes cuando se necesitan más de una pareja de paréntesis, sino siempre paréntesis (como puedes apreciar en el quinto ejemplo).

Como ha quedado patente, la prioridad de los operadores es de vital importancia para tener los resultados deseados. Existe un conjunto de reglas que define la prioridad de cada tipo de operador. Las verás un poco más adelante.

Por tanto se puede concluir que una **expresión** es una secuencia determinada de operandos y operadores escrita con unas reglas de sintaxis determinadas por el lenguaje de programación.

4.1 Operadores

Ya viste en el tema anterior, entre los elementos del pseudocódigo, una serie de operadores, ahora puedes ampliar tus conocimientos un poco más sobre ellos. Existen varios tipos de clasificaciones de los operadores. Por ejemplo, podemos clasificarlos atendiendo al número de operandos que tienen (operadores unarios, operadores binarios, operadores ternarios, ...). Esta clasificación es fundamental a la hora de aplicar la sintaxis para desarrollar expresiones que contengan operadores, o sea, para que las expresiones estén bien formadas. También podemos clasificarlos por el tipo de operaciones que realizan, encontrándonos los siguientes:

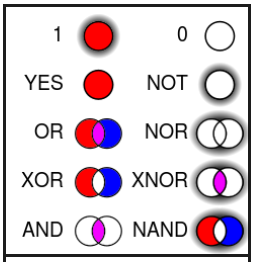


Imagen en Wikimedia de HereToHelp bajo licencia GNU

- **Operadores aritméticos**: Son los que se usan para realizar operaciones matemáticas. Los más usuales son los de la tabla siguiente:

OPERADOR	SIGNIFICADO
+	Suma
-	Resta
*	Multiplicación
/	División
^	Potencia. Puede variar de un lenguaje a otro, incluso en algunos está en forma de función.
DIV	División entera. Suele variar entre los distintos lenguajes de programación.
MOD	Resto de la división entera. También este operador suele tener distinta simbología según el lenguaje.

- **Operadores relacionales**: Se utilizan para formar expresiones que dan como resultado dos valores posibles, o verdadero o falso. Es decir, o se cumple la expresión o no se cumple. La lista de operadores relacionales aparece en la tabla siguiente:

OPERADOR	SIGNIFICADO
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
=	Igual. En algunos lenguajes, se utiliza como operador el == (para diferenciarlo del operador de asignación que verás más abajo)
<>	Distinto. Hay lenguajes que usan el símbolo !=

- **Operadores lógicos**: Son aquellos operadores que permiten unir en una sola expresión varias expresiones que contienen operadores relacionales, y por tanto, el resultado final de la expresión global será verdadero o falso. En la siguiente tabla tienes los fundamentales:

OPERADOR	SIGNIFICADO
OR (O)	Suma lógica o disyunción. Hay lenguajes que usan el símbolo de la barra vertical, simple (|) o doble (||)
AND (Y)	Producto lógico o conjunción. En algunos lenguajes se usa el ampersand & o el doble ampersand (&&)
NOT (NO)	Negación. También este operador puede variar dependiendo del lenguaje, pudiéndo ser ! o incluso ~

- **Operadores alfanuméricos o de cadenas**: Estos operadores son los que posibilitan realizar operaciones sobre los textos. La mayoría de lenguajes suelen aportar estos operadores en forma de funciones, ya irás familiarizándote con este concepto más adelante. Una de las operaciones que más se utiliza de este tipo es la de concatenación de dos textos (la unión de dos textos en uno solo). En muchos lenguajes, el operador de concatenación coincide con el de la suma aritmética (el signo +), en otros se usa otro tipo de símbolos (el ampersand, la barra vertical doble, el punto, etc) y hay lenguajes que aportan el operador en forma de función, al igual que el resto de operadores de cadenas.
- **Operador de asignación**: Es el que sirve para asignar valores a variables y constantes. Ya te has tropezado varias veces con este operador en el tema anterior. En pseudocódigo se utiliza el símbolo de la flecha horizontal apuntando hacia la izquierda (←), en la mayoría de lenguajes de programación se usa el signo igual (=).

Para saber más

Operadores lógicos: Los operadores lógicos (and, or, not, ...) son los que permiten encadenar expresiones relacionales en una sola, siguiendo las reglas que marcan el Álgebra de Boole. Buscar información al respecto te vendrá muy bien para ampliar tus conocimientos sobre las expresiones relacionales.

Si necesitas conocer los operadores que el entorno PSeInt utiliza, puedes consultarlos aquí.

4.2 Tipos de Expresiones

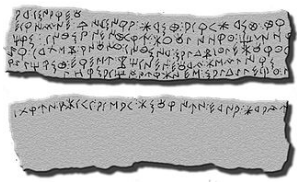


Imagen en Wikimedia de Papix bajo licencia GNU

Dependiendo de los tipos de operadores que incluyen, las expresiones pueden ser de cinco tipos: aritméticas, lógicas, relacionales, alfanuméricas y de asignación. A continuación puedes ver algunos ejemplos de cada uno de ellos:

- **Expresiones aritméticas**: son aquellas que utilizan operadores aritméticos y como operandos tienen datos numéricos. Ejemplos:

EXPRESIÓN	RESULTADO
20+30	50
35-20	15
4*5	20
18/6	3
2^4	16
17 DIV 5	3
17 MOD 5	2

- **Expresiones relacionales**: Son las expresiones en las que aparecen los operadores relacionales. El resultado de expresiones relacionales siempre será uno de dos valores posibles, o verdadero o falso. Ejemplos:

EXPRESIÓN	RESULTADO
3<6	VERDADERO
3<=2	FALSO
10>18	FALSO
10>=10	VERDADERO
5=6	FALSO
5<>6	VERDADERO

- **Expresiones lógicas**: son aquellas que usan exclusivamente operadores lógicos. El resultado de una expresión lógica siempre es verdadero o falso. El operador Y hará que la expresión sea verdadera sólo y exclusivamente cuando los dos operandos sean verdaderos. Una expresión con el operador O será verdadera cuando alguno de los dos operandos, o los dos, sean verdaderos. El operador NO negará la expresión, es decir, que el resultado final será verdadero cuando el operando sea falso, o viceversa. Ejemplos:

EXPRESIÓN	RESULTADO
10<20 y 40>25	**Verdadero**, porque se cumplen las dos condiciones, es decir 10 es menor que 20 y 40 es mayor que 25.
10<20 y 40<25	**Falso**, ya que no se cumplen las dos condiciones, es decir 10 si es menor que 20 pero 40 es mayor que 25, no cumpliendose la segunda condición que invalida toda la expresión.
10<20 o 40>25	**Verdadero**, debido a que se cumplen las dos condiciones, es decir 10 es menor que 20 y 40 es mayor que 25, pero con que sólo una de ellas fuera cierta ya su resultado sería verdadero.

10<20 o 40<25	**Verdadero**, porque se cumple una de las dos condiciones, es decir 10 si es menor que 20, aunque no se cumpla la segunda condición.
20<10 o 40<25	**Falso**, porque no se cumple ninguna de las dos condiciones.
NO(40>25)	**Falso**, ya que al evaluar la expresión del interior del paréntesis en primer lugar (40>25) el resultado es verdadero, y al negar lo verdadero conseguimos un falso (algo NO VERDADERO es FALSO).
NO(40<25)	**Verdadero**, pues el resultado de la expresión del interior de los paréntesis es falso, y al negar lo falso conseguimos un verdadero (algo NO FALSO es VERDADERO).

- **Expresiones alfanuméricas**: En las expresiones de carácter no existen operadores y en las expresiones de cadena sólo existe uno, el +, concatenación, que lo que hace es unir cadenas. Ejemplos:

EXPRESIÓN	RESULTADO
"L"+"A"	"LA"
"EL"+"PROGRAMADOR"	"ELPROGRAMADOR"
"EL BUEN "+"PROGRAMADOR"	"EL BUEN PROGRAMADOR"

- **Expresiones de asignación**: Estas expresiones ya las conoces, se utilizan para asignarle valores a las variables o constantes. Tienen dos operandos: el operando de la izquierda del operador será siempre un identificador que corresponderá a una variable o a una constante, mientras que el operando de la derecha del operador puede ser un valor fijo, una variable, una constante o una expresión. Ambos operandos tienen que ser del mismo tipo de dato.

EXPRESIÓN	RESULTADO
nombre ← "Belén"	A la variable nombre se le asigna la cadena "Belén"
base ← 5	A la variable base se le asigna un 5
area ← base * altura / 2	A la variable area se le asigna el resultado de calcular el área de un triángulo

Curiosidad

Imagen en Flickr de Pete Simon. Licencia CC

Hasta ahora, hemos visto expresiones relacionales en las que se comparaban números o expresiones numéricas, sin embargo, los operadores relacionales pueden comparar expresiones de otros tipos, con el único requisito de que sean del mismo tipo. Así, pueden comparar caracteres o cadenas de caracteres.

Para caracteres y cadenas rige el código ASCII (o UNICODE) que usa el ordenador de tal forma que un carácter será menor que otro si su código ASCII es menor que el del otro carácter. En estas

de caracteres.

4.3 Prioridad de los operadores

Imagen de creación propia bajo licencia CC

Cuando se realizan expresiones, en cualquier lenguaje, incluyendo pseudocódigo, es habitual mezclar en ellas diferentes operadores, incluso de distintos tipos (aritméticos, relacionales, lógicos o de cadena).

Ya viste que en el caso de los numéricos las operaciones entre paréntesis tienen prioridad. Además, ya sabes que se pueden anidar distintas secuencias de paréntesis como has apreciado en ejemplos anteriores. En esos casos se prioriza el más interno al más externo y después de izquierda a derecha. Por tanto en ((3*5)+20)*4 lo primero es multiplicar 3 por 5, despues al resultado que es 15 sumar 20, y este resultado que es 35 multiplicarlo por 4. Además el orden de operación alterado por el uso de paréntesis altera lógicamente el resultado final (lo vemos a continuación con un ejemplo).

Cuando se utilizan operadores mezclados, la prioridad es la siguiente:

1. Paréntesis (si hay varias parejas de paréntesis, de los más internos a los más externos).
2. Aritméticos (en el mismo orden que en matemáticas): ^, *, /, +, -
3. Concatenación.
4. Relacionales.
5. Lógicos.

Veamos algunos ejemplos:

EXPRESIÓN	RESULTADO
3*5<8 O 4+6>7	El resultado de esta expresión es **verdadero**, porque si primero ejecutamos los operadores aritméticos, la expresión quedaría como 15<8 o 10>7. Ya que para el operador lógico O si cualquiera de las comparaciones es verdadero el resultado es verdadero se cumple pues 15 no es menor que 8 pero 10 si es mayor que 7. Primero han intervenido los aritméticos, después los relacionales y por último el lógico.
(2*12>9+8) Y 20<18	El resultado será **falso**, pues sólo se cumple que 24 es mayor que 17 pero no que 20 sea menor que 18. Para el operador lógico Y, sólo se obtiene resultado verdadero si todas las condiciones se cumplen.
(50 MOD 4)*3	El resultado es **6**, pues el dividendo es 50, el divisor 4, el cociente 12 y el resto 2. Al multiplicar 2 por 3 nos da 6.
50 MOD (4*3)	El resultado es **2**, pues el dividendo es 50, el divisor 12, el cociente 4 y el resto 2.

Se puede corroborar que la alteración del orden de ejecución impuesto por los paréntesis, colocados en distinto lugar, da resultados diferentes como habíamos indicado anteriormente.

5. Comentarios

Imagen en Pixabay de fancycrave1 bajo licencia CC0

Cuando se construye un programa, el programador lo implementa traduciendo su algoritmo al lenguaje de programación que va a utilizar para ello. El resultado es una serie de líneas de código, el llamado código fuente, regidas por una sintaxis concreta y en las que se utiliza una simbología específica, propias del lenguaje de programación elegido.

Pues bien, cuando el programa resultante es pequeño y tiene pocas líneas de código, suele ser fácil de entender, incluso por otros programadores. En cambio, cuando las líneas de código aumentan, también lo hace la dificultad para entenderlo, volviéndose un código fuente oscuro y tenebroso, en ocasiones hasta para el propio autor del programa. A medida que se va construyendo el programa, el propio programador debe ser consciente de tal hecho, y debe pensar en todo momento que posiblemente en un futuro, a veces lejano, necesitará volver a retomar el código fuente que está escribiendo para realizar alguna modificación, alguna corrección o simplemente alguna mejora. Es por ello, por lo que todo lenguaje de programación ofrece al programador una herramienta fundamental para paliar este problema: el uso de **comentarios**.

Los comentarios son anotaciones que el programador incorpora a su código fuente para hacerlo más entendible. Estas líneas son ignoradas por el ordenador cuando se ejecuta el programa, por lo que son inocuas para la obtención del resultado final.

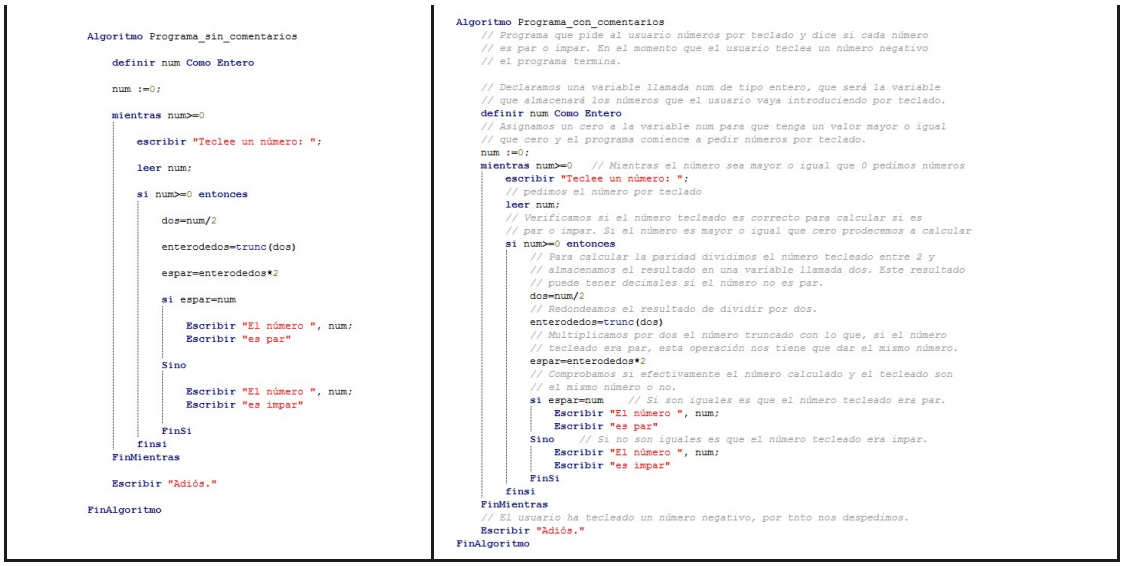
Cada lenguaje de programación establece una simbología para marcar los comentarios dentro del código fuente. Por ejemplo, en pseudocódigo suele usarse una doble línea diagonal, es decir, la barra de dividir dos veces seguidas (algo así //). Cuando se quiere escribir un comentario, se comenzará escribiendo las dos barras seguidas del comentario, el ordenador sabrá que lo que aparece a la derecha de las dos barras es un comentario y lo ignorará en la ejecución. También esta simbología es usada en algunos lenguajes de programación, aunque hay más variantes.

En la mayoría de lenguajes **existen dos posibilidades** para ello:

a. Se puede escribir una única línea de comentario (toda la línea es un comentario), o añadir el comentario al final de una misma línea de código fuente. En estos casos el comentario empieza después de un carácter o conjunto de caracteres que el lenguaje especifica.
b. Se puede escribir un conjunto de líneas de comentarios consecutivas. Para ello habrá un carácter (o varios) para indicar dónde empiezan las líneas de comentarios y otro carácter (o varios) para indicar dónde terminan. Todo lo que esté entre el carácter (o los caracteres) de comienzo y el de fin es tomado como comentario e ignorado al ejecutar el programa.

A continuación vas a ver tú mismo dos ejemplos del mismo programa escrito en pseudocódigo, una primera versión sin incluir comentarios y una segunda versión con los pertinentes comentarios que el programador ha estimado oportuno intercalar para mejorar la compresión del código fuente:

CÓDIGO FUENTE SIN COMENTARIOS	CÓDIGO FUENTE CON COMENTARIOS



```
Algoritmo Programa_sin_comentarios

    definir num Como Entero

    num :=0;

    mientras num>=0

        escribir "Teclee un número: ";

        leer num;

        si num>=0 entonces

            dos=num/2

            enterodedos=trunc(dos)

            espar=enterodedos*2

            si espar=num

                Escribir "El número ", num;
                Escribir "es par"

            Sino

                Escribir "El número ", num;
                Escribir "es impar"

            FinSi
        finsi
    FinMientras

    Escribir "Adiós."

FinAlgoritmo
```

```
Algoritmo Programa_con_comentarios
    // Programa que pide al usuario números por teclado y dice si cada número
    // es par o impar. En el momento que el usuario teclea un número negativo
    // el programa termina.

    // Declaramos una variable llamada num de tipo entero, que será la variable
    // que almacenará los números que el usuario vaya introduciendo por teclado.
    definir num Como Entero
    // Asignamos un cero a la variable num para que tenga un valor mayor o igual
    // que cero y el programa comience a pedir números por teclado.
    num :=0;
    mientras num>=0   // Mientras el número sea mayor o igual que 0 pedimos números
        escribir "Teclee un número: ";
        // pedimos el número por teclado
        leer num;
        // Verificamos si el número tecleado es correcto para calcular si es
        // par o impar. Si el número es mayor o igual que cero procedemos a calcular
        si num>=0 entonces
            // Para calcular la paridad dividimos el número tecleado entre 2 y
            // almacenamos el resultado en una variable llamada dos. Este resultado
            // puede tener decimales si el número no es par.
            dos=num/2
            // Redondeamos el resultado de dividir por dos.
            enterodedos=trunc(dos)
            // Multiplicamos por dos el número truncado con lo que, si el número
            // tecleado era par, esta operación nos tiene que dar el mismo número.
            espar=enterodedos*2
            // Comprobamos si efectivamente el número calculado y el tecleado son
            // el mismo número o no.
            si espar=num    // Si son iguales es que el número tecleado era par.
                Escribir "El número ", num;
                Escribir "es par"
            Sino    // Si no son iguales es que el número tecleado era impar.
                Escribir "El número ", num;
                Escribir "es impar"
            FinSi
        finsi
    FinMientras
    // El usuario ha tecleado un número negativo, por tnto nos despedimos.
    Escribir "Adiós."
FinAlgoritmo
```

Como puedes apreciar, aun siendo el programa pequeño, se entiende mucho mejor si se incluyen ciertos comentarios entre las líneas del código fuente. Imagina lo que ocurre si no se usan comentarios y se trata de un programa de miles de líneas de código...

Comprueba lo aprendido

Son tipos de datos básicos de pseudocódigos:

- Las cadenas y los símbolos de los diagrmas de flujo.
- Los caracteres.
- Los caracteres y los símbolos de pseudocódigo.

Incorrecto

Opción correcta

Incorrecto

Solution

1. Incorrecto
2. Opción correcta
3. Incorrecto

Los datos básicos lógicos:

- Admiten tres valores resultado: verdadero, neutro y falso.
- Admiten tres valores resultado: positivo, cero y negativo.
- Sólo admiten verdadero o falso.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

De los conjuntos numéricos Z y R, sabemos que:

Los dos son conjuntos infinitos de términos.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Un ordenador que sea muy potente:

Puede manejar sin problemas el conjunto Z completamente, pero no R íntegramente.

Puede manejar los dos íntegramente si es lo bastante potente.

No puede manejarlos completamente, sólo un subconjunto de ellos.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Las cadenas "La Odisea" y "La_Odisea"

Son iguales y tienen el mismo número de caracteres.

Son diferentes y tienen el mismo número de caracteres.

Son diferentes y no tienen el mismo número de caracteres.

Incorrecto

Opción correcta

Incorrecto

Solution

3. Incorrecto

Las cadenas "gato", "Gato" y "GATO":

- La primera y la tercera Son equivalentes para el ordenador.
- Las dos primeras son iguales.
- Son las tres diferentes.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Una variable:

- Es un espacio de la memoria que admite cambios de diferentes tipos de datos.
- Es un espacio de la memoria que admite cambios en los valores de los datos que almacena.
- Es un dato.

Incorrecto

Opción correcta

Incorrecto

Solution

1. Incorrecto
2. Opción correcta
3. Incorrecto

Una constante:

- No puede variar durante la ejecución de un programa.
- Puede variar si la convertimos en variable.
- No es un dato.

Incorrecto

Incorrecto

Solution

1. Opción correcta
2. Incorrecto
3. Incorrecto

Una constante:

- Puede ser de tipo entero y lógico.
- Puede ser entera o real, pero nunca lógica.
- No puede ser carácter.

Opción correcta

Incorrecto

Incorrecto

Solution

1. Opción correcta
2. Incorrecto
3. Incorrecto

Los operadores:

- Interrelacionan con los operandos para formar las expresiones.
- No hacen falta para formar las expresiones.
- Puede existir una expresión con operandos sin incluir ningún operador.

Opción correcta

Incorrecto

Incorrecto

Solution

1. Opción correcta
2. Incorrecto
3. Incorrecto

- El tipo de dato almacenado restringe el tipo de operadores que se le pueden aplicar.
- Es indiferente el tipo de dato para operadores complejos.

Incorrecto

Opción correcta

Incorrecto

Solution

1. Incorrecto
2. Opción correcta
3. Incorrecto

Los paréntesis en las expresiones:

- Sólo indican prioridad en las operaciones numéricas.
- En cualquier caso el operador multiplicación tiene prioridad.
- Indican prioridad en el orden de ejecución.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Existen expresiones:

- Aritméticas y lógicas.
- Aritméticas y literales.
- Simples y compuestas.

Opción correcta

Incorrecto

Incorrecto

2. Incorrecto
3. Incorrecto

La concatenación es:

- Un tipo básico de dato.
- Un operador de cadena.
- Una cadena de operadores.

Incorrecto

Opción correcta

Incorrecto

Solution

1. Incorrecto
2. Opción correcta
3. Incorrecto

El operador <> es:

- Aritmético.
- De cadena.
- Relacional.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Los operadores relacionales:

- Comparan datos del mismo tipo.
- Comparan datos de diferente tipo.
- No comparan datos.

Incorrecto

Incorrecto

Solution

1. Opción correcta
2. Incorrecto
3. Incorrecto

Comprueba lo aprendido

Indica verdadero o falso en las siguientes expresiones:

Las mayúsculas y minúsculas son indiferentes para comparaciones relacionales.

Verdadero Falso

Falso

45>20.

Verdadero Falso

Verdadero

NO(45>20).

Verdadero Falso

Falso

45<20.

Verdadero Falso

Falso

NO(45<20).

Verdadero

((2+3)*4)+5=25

Verdadero Falso

Verdadero

(2+(3*4))+5=25

Verdadero Falso

Falso

Las expresiones de los dos apartados anteriores no son válidas pues no se pueden anidar paréntesis.

Verdadero Falso

Falso

En caso de paréntesis anidados tienen prioridad los internos frente a los externos.

Verdadero Falso

Verdadero

En caso de paréntesis anidados tienen prioridad los externos frente a los internos.

Verdadero Falso

Falso

En expresiones con operadores de distinto tipo tienen prioridad los relacionales.

Verdadero Falso

Falso

Los primeros operadores que se aplican son los aritméticos.

Verdadero Falso

Verdadero

Verdadero Falso

Falso

Los paréntesis influyen en el resultado de una expresión, salvo si el procesador es potente.

Verdadero Falso

Falso

Son operadores lógicos: Y, O, NI.

Verdadero Falso

Falso

Una expresión con varios operandos seguidos sin operadores puede ser válida.

Verdadero Falso

Falso

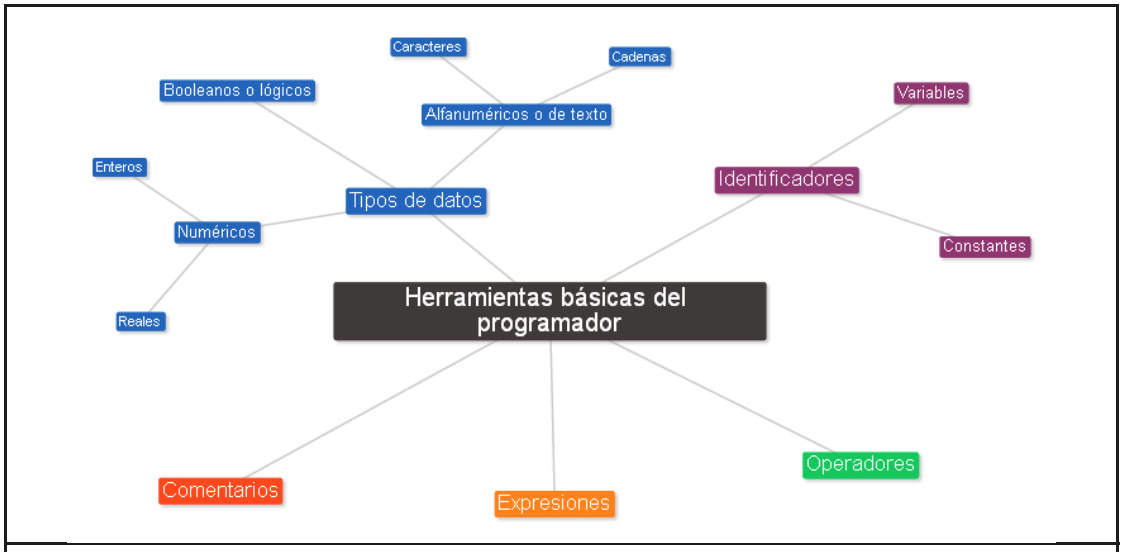


Imagen de creación propia con licencia Creative Commons

Descargar PDF

Aviso Legal

El presente texto (en adelante, el "**Aviso Legal**") regula el acceso y el uso de los contenidos desde los que se enlaza. La utilización de estos contenidos atribuye la condición de usuario del mismo (en adelante, el "**Usuario**") e implica la aceptación plena y sin reservas de todas y cada una de las disposiciones incluidas en este Aviso Legal publicado en el momento de acceso al sitio web. Tal y como se explica más adelante, la autoría de estos materiales corresponde a un trabajo de la **Comunidad Autónoma Andaluza, Consejería de Educación y Deporte (en adelante Consejería de Educación y Deporte**).

Con el fin de mejorar las prestaciones de los contenidos ofrecidos, la Consejería de Educación y Deporte se reserva el derecho, en cualquier momento, de forma unilateral y sin previa notificación al usuario, a modificar, ampliar o suspender temporalmente la presentación, configuración especificaciones técnicas y servicios del sitio web que da soporte a los contenidos educativos objeto del presente Aviso Legal. En consecuencia, se recomienda al Usuario que lea atentamente el presente Aviso Legal en el momento que acceda al referido sitio web, ya que dicho Aviso puede ser modificado en cualquier momento, de conformidad con lo expuesto anteriormente.

Régimen de Propiedad Intelectual e Industrial sobre los contenidos del sitio web.

Imagen corporativa. Todas las marcas, logotipos o signos distintivos de cualquier clase relacionados con la imagen corporativa de la Consejería de Educación y Deporte que ofrece e