

Desarrollo de contenido

Unidad 1

La informática y los algoritmos

Fundamentos de Programación

Presentación general del curso

Desde el ábaco, en la época antes de cristo, hasta las poderosas máquinas de cómputo de nuestros días, el hombre ha tenido la imperativa necesidad de crear y hacer uso de herramientas que le faciliten una mejor comprensión e interacción con su entorno, estos avances tecnológicos le han permitido aumentar la productividad, optimizar los recursos, automatizar tareas repetitivas, realizar cálculos de alta complejidad, y en general, desarrollar de una manera más eficiente cualquier tipo de actividad tanto en el hogar como en la industria.

La ingeniería mecatrónica representa la sinergia entre la mecánica, la electrónica, el control y la programación. La mecatrónica cuenta con un lugar privilegiado y protagónico en el conocimiento e integración de nuevas tecnologías que dinamizan el desarrollo económico en los países, como es el caso de la robótica, la inteligencia artificial, el internet de las cosas, entre otras de la industria 4.0, todas ellas con un elemento clave en común, la programación.

La asignatura fundamentos de programación facilita ese primer acercamiento a lo que son los sistemas informáticos, su evolución y la forma en la que se integran el hardware y el software en un sistema computacional. En este curso el estudiante se enfrentará a problemas de programación, a analizarlos e interpretarlos, desde su concepción, hasta el planteamiento de una solución lógica debidamente estructurada y representada en formato algorítmico. Dentro de las habilidades que debe desarrollar el estudiante, están la correcta interpretación y uso de los diferentes tipos de datos, operadores y controladores de flujo, además estará en capacidad de representar las soluciones en forma de pseudocódigo, diagrama de flujo o como un algoritmo básico codificado en un lenguaje de programación específico.

Objetivos del curso / Objetivos de aprendizaje

- **Objetivo general/Objetivo de la asignatura:**

Resolver problemas de programación mediante algoritmos codificados que brinden soluciones estructuradas que den cuenta del uso apropiado de los diferentes tipos de datos, operadores y controladores de flujo.

- **Objetivos específicos/Resultado de aprendizaje de la asignatura**
- Analizar problemas básicos de programación, desde su interpretación lógica hasta el planteamiento de una solución algorítmica representada con pseudocódigos y diagramas de flujo.
- Aplicar e interpretar correctamente la estructura de los datos, los operadores y los controladores de flujo que forman parte de un algoritmo.
- Construir soluciones algorítmicas codificadas en un lenguaje de programación, aplicando debidamente la estructura y sintaxis de sus elementos.

Pregunta orientadora

Un robot humanoide es un sistema mecatrónico en el que se evidencia la interacción de las cuatro disciplinas de la mecatrónica. **La electrónica** la compone la sensórica y los circuitos electrónicos necesarios para leer el entorno y generar señales eléctricas. **La mecánica** es el conjunto de actuadores que proporcionan movimiento al robot. **El control** hace referencia a las técnicas utilizadas para garantizar la estabilidad y exactitud en su operación. Por último, con **la programación** se automatiza el desempeño a través de dispositivos programables que alojan algoritmos codificados en un lenguaje de programación. La eficiencia en el desempeño dependerá de la estructura lógica del código que lo controla, para esto es fundamental interpretar correctamente la problemática a resolver, por otro lado, se debe comprender la forma en la que se relacionan las entradas (sensores) y las salidas del sistema (actuadores), para realizar operaciones que hagan que el robot tome decisiones más acertadas (procesamiento).

Al finalizar el curso el estudiante estará en capacidad de dar solución a la siguiente pregunta:

¿Cómo resolver problemas básicos de programación, desde su interpretación, hasta la representación algorítmica codificada, haciendo uso apropiado de los diferentes tipos de datos, operadores y controladores de flujo?

Mapa del curso



Fundamentos de Programación

UNIDAD

1

La informática y los algoritmos

Antecedentes históricos

- Evolución de los sistemas informáticos
- ¿Qué son los lenguajes de programación?
- El computador y los sistemas de numeración

Operadores, datos y estructuras de control de flujo

- Operadores y expresiones aritméticas, relacionales y lógicas
- Datos numéricos, lógicos y de texto
- Estructuras de control de flujo: secuencial, de decisión, de repetición y de salto
- Constantes, variables, contadores y acumuladores
- Notación algorítmica de expresiones aritméticas

Los algoritmos

- Estructura de un algoritmo
- Representación: relato, pseudocódigo y diagrama de flujo
- Análisis de algoritmos
- Herramienta para la elaboración de algoritmos
- Resolución de problemas con algoritmos
- Ejercicios prácticos



Fundamentos de Programación

UNIDAD
2

Algoritmos codificados

Sintaxis del lenguaje de programación

- Datos, operadores, expresiones y estructuras de control de flujo

Codificación del algoritmo en el lenguaje de programación

Entorno de desarrollo del lenguaje de programación

- El intérprete de comandos

- Ejercicios prácticos

UNIDAD
3

Arreglos

Funciones: estructuras y llamados

Conceptos y uso de vectores

Conceptos y uso de matrices

Ejercicios prácticos

Cronograma de actividades de la asignatura o módulo

Actividad de aprendizaje*	Evidencia de aprendizaje**	Semana***	Ponderación
AA. Acercamiento a las temáticas	EA. Conocimientos previos	Semana 1	0%
AA1. La informática y los algoritmos	EA1. Proyecto integrador parte uno	Semana 3	20%
AA2. Algoritmos codificados	EA2. Proyecto integrador parte dos	Semana 5	25%
AA3. Arreglos	EA3. Proyecto integrador parte tres	Semana 7	25%
EA4 Proyecto integrador - Entrega final		Semana 8	30%
Total			100%

Unidad 1. La informática y los algoritmos

Introducción a la Unidad 1

¿Alguna vez te has preguntado cómo ha sido posible que la humanidad haya evolucionado tanto desde la invención del ábaco hasta la creación de dispositivos como los teléfonos inteligentes, provistos de procesadores capaces de realizar coordinadamente múltiples tareas como llamadas, reproducción de música, tomar fotografías, usar redes sociales, entre muchas otras aplicaciones? Bueno, pues todo esto es posible gracias a la combinación entre hardware y software, lo primero tiene que ver con todo lo que es físico en un dispositivo, y lo segundo, con todo lo que no lo es, pero que gobierna al hardware a través de algoritmos, que es precisamente lo que se estudiará en esta primera unidad.

Objetivos de aprendizaje de la Unidad 1

- Comprender los conceptos básicos de los sistemas informáticos.
- Interpretar un problema básico de programación identificando correctamente las entradas, salidas y operaciones necesarias para solucionarlo.
- Representar una solución algorítmica en formato de pseudocódigo o diagrama de flujo.
- Aplicar e interpretar correctamente la estructura de los datos, los operadores y los controladores de flujo que forman parte de un algoritmo.

Cronograma de actividades 1

Cronograma de actividades Unidad 1			
Actividad de aprendizaje*	Evidencia de aprendizaje**	Semana***	Ponderación
AA. Actividad de conocimientos previos	EA. Foro	Semana 1	0%
AA1. La informática y los algoritmos	EA1. proyecto integrador parte uno	Semana 3	20%
Total			20%

Unidad 1. Actividad de aprendizaje 1: Antecedentes históricos

Inicia el desarrollo de los temas de la actividad de aprendizaje

¿Qué temas estudiarás en la Unidad 1?

Antecedentes históricos

- Evolución de los sistemas informáticos
- ¿Qué son los lenguajes de programación?
- El computador y los sistemas de numeración

Operadores, datos y estructuras de control de flujo

- Operadores y expresiones aritméticas, relacionales y lógicas
- Datos numéricos, lógicos y de texto
- Estructuras de control de flujo: secuencial, de decisión, de repetición y de salto
- Constantes, variables, contadores y acumuladores
- Notación algorítmica de expresiones aritméticas

Los algoritmos

- Estructura de un algoritmo
- Representación: pseudocódigo y diagrama de flujo
- Análisis de algoritmos
- Herramienta para la elaboración de algoritmos
- Resolución de problemas con algoritmos
- Ejercicios prácticos

1. Antecedentes históricos

1.1. Evolución de los sistemas informáticos

La necesidad inminente del ser humano por construir herramientas que le permitieran realizar cálculos de manera automática, como se aprecia en la Figura 1, lo llevó a dar sus primeros pasos con la construcción del ábaco en el periodo antes de cristo (A.C.), más adelante, en 1642, Blaise Pascal creó la primera calculadora, seguido de Goffried Liebnits, quien desarrolló la primera máquina de multiplicar.

En 1820, Thomas Delaware, inventó la calculadora de uso industrial llamada aritmómetro, y en 1822, el ingeniero y matemático Charles Babbage, conocido también como el padre de la computación, concibe una máquina diferencial que funcionaba con un motor a vapor, capaz de trabajar con logaritmos, funciones trigonométricas y polinomios.

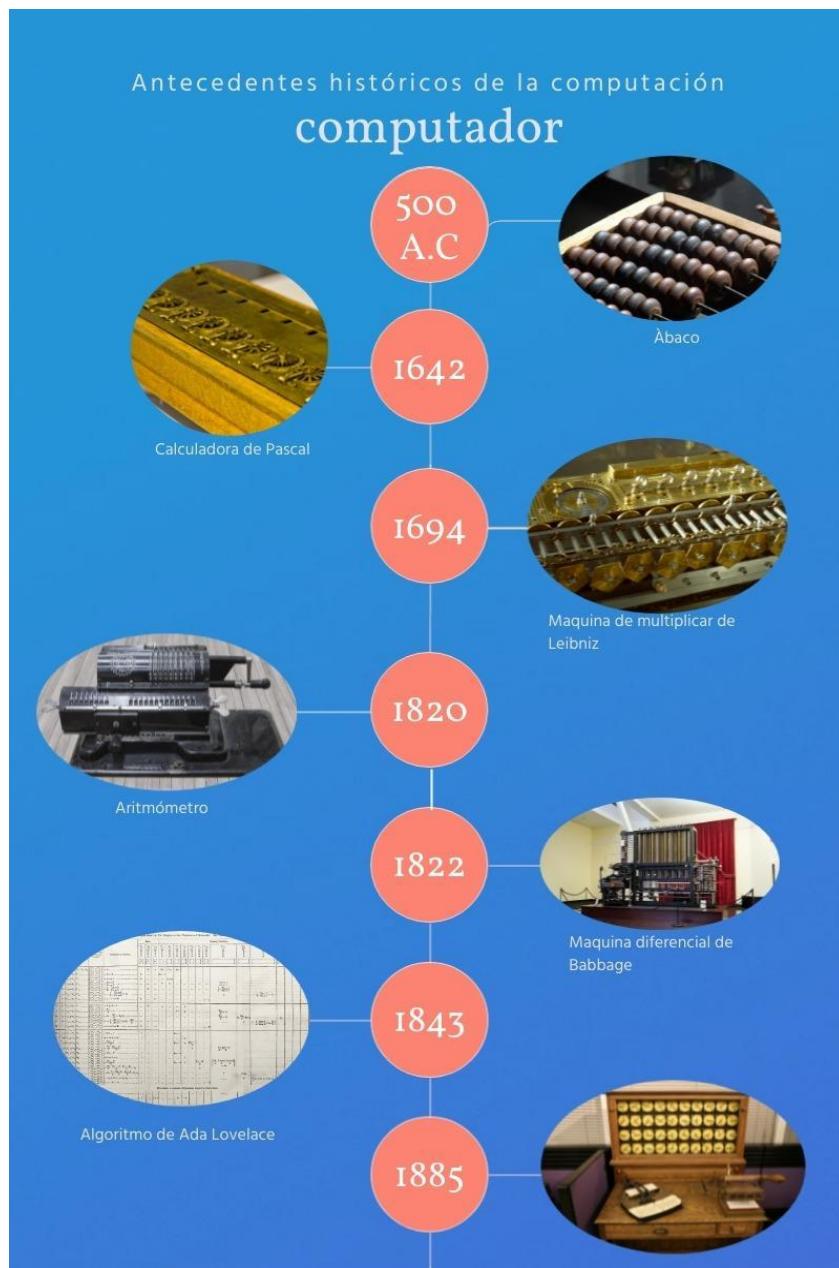
En 1843, aparece la primera mujer en la historia de la computación, Ada Lovelace, quien escribió un artículo en el que incluyó el que es considerado como el primer algoritmo para ser ejecutado por una máquina, lo que la catapultó como la primera programadora en la historia. El algoritmo calculaba los valores de los números de Bernoulli. En 1885, Germán Hollerith inventó el tabulador, en él usó tarjetas perforadas para procesar información estadística. En 1936, Alan Turing presenta lo que se llamó la máquina de Turing, que no era otra cosa que un algoritmo que representaba un modelo matemático autónomo capaz de resolver un problema matemático. Sus ideas dieron pie para la concepción de lo que es la computadora moderna, además su gran ingenio permitió, durante la segunda guerra mundial, que los ingleses descifraron los códigos secretos enviados por los alemanes.

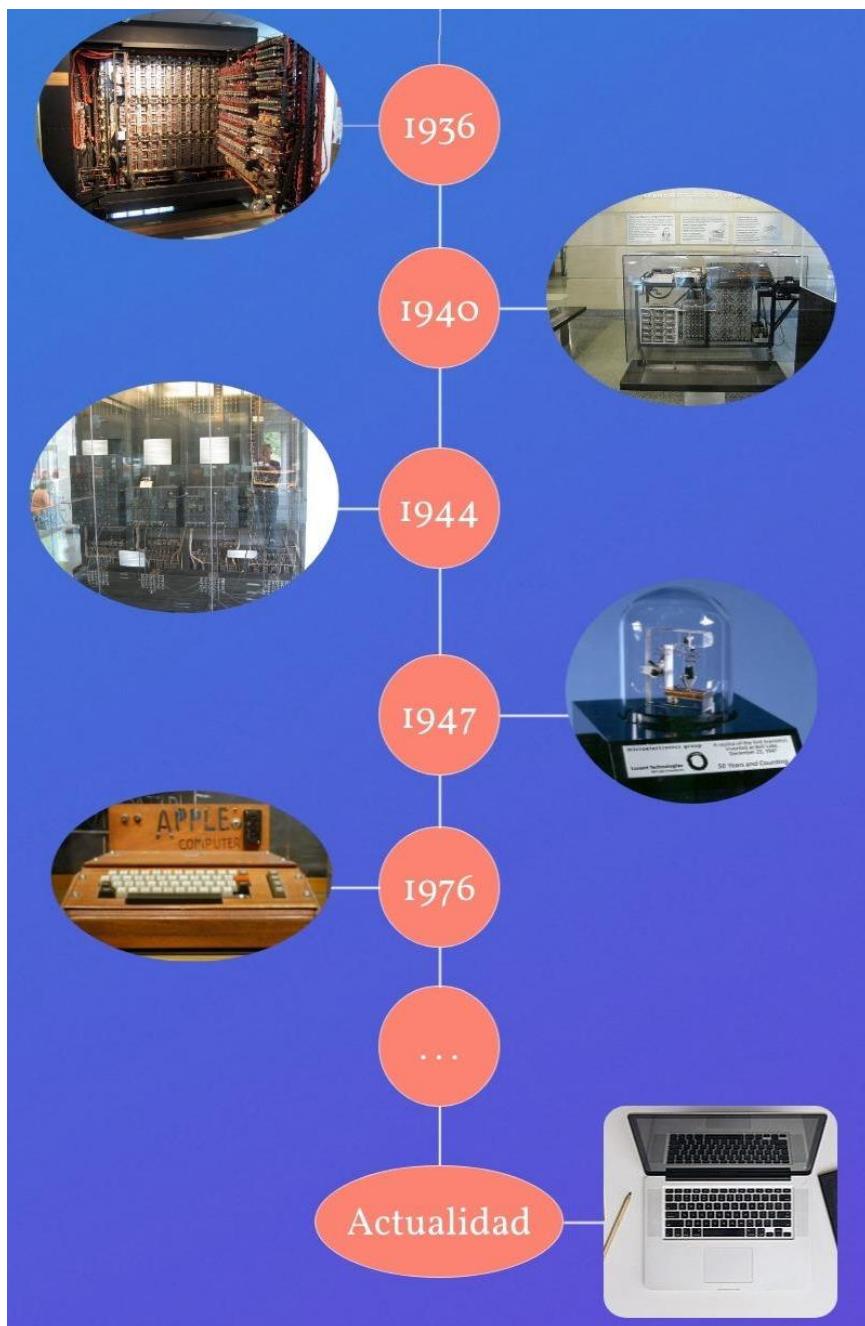
La empresa Hewlett-Packard es fundada por Bill Hewlett y David Packard en Palo Alto, el nombre de dicha empresa es creado con base a sus respectivos apellidos. Posteriormente, Atanasoff y su estudiante de posgrado, Clifford Berry, diseñan una computadora capaz de resolver simultáneamente 29 ecuaciones, además fue el primer ordenador capaz de almacenar información en su memoria principal. Luego, los profesores de la Universidad de Pensilvania John Mauchly y J. Presper Eckert, construyeron el primer Computador e Integrador Numérico Electrónico, más conocido como ENIAC (Electronic Numerical Integrator And Computer), considerado como el padre de la era de los ordenadores digitales, y construido en su totalidad por 18000 tubos de vacío. Más tarde, la Universidad de Pensilvania recibió fondos de la oficina de Censo para construir la UNIVAC, la primera computadora comercial para negocios y aplicaciones gubernamentales.

Mas adelante, William Shockley, John Bardeen y Walter Brattain, de Bell Laboratories, inventan el transistor, un dispositivo de material semiconductor que sustituyó a los tubos de vacío, con lo que se optimizó espacio y se redujo considerablemente el consumo de energía en la implementación de los computadores. En los años siguientes Gracia Hopper desarrolla el primer lenguaje de programación orientado a los negocios, llamado COBOL (COmmon Business Oriented Language). Posteriormente, IBM desarrolla el lenguaje de programación conocido como FORTRAN (Formula Translating System). Luego Jack Kilby y Robert Noyce elaboran el primer circuito integrado o chip para ordenadores. En 1964, Douglas Engelbart presenta su prototipo de ordenador moderno, compuesto por un mouse y una interfaz gráfica de usuario (GUI – Graphic User Interface), este prototipo revolucionó la industria, e hizo que las computadoras evolucionaran de un uso netamente matemático y científico, a ser una máquina accesible para el público general. En 1969, un

grupo de desarrolladores de laboratorios Bell de AT&T desarrollan UNIX, un sistema operativo portable, multitarea y multiusuario.

Figura 1. Antecedentes históricos de la computación





¿Sabías qué?



La primera programadora en todo el mundo fue una mujer: Ada Lovelace, quien nació en 1815 y publicó el primer algoritmo destinado a ser ejecutado por una máquina.

1.2. El computador y los sistemas de numeración

¿Qué es un computador?

Un computador es una máquina programable capaz de leer, almacenar y procesar información. El computador se compone de dos elementos básicos: **el hardware y el software**. En la *Figura 2* se observan las partes que componen un computador de escritorio convencional.

Figura 2. El computador



Fuente: Pixabay. Imagen de Clker-Free-Vector-Images en Pixabay

<https://pixabay.com/es/vectors/puesto-de-trabajo-computadora-303940/>

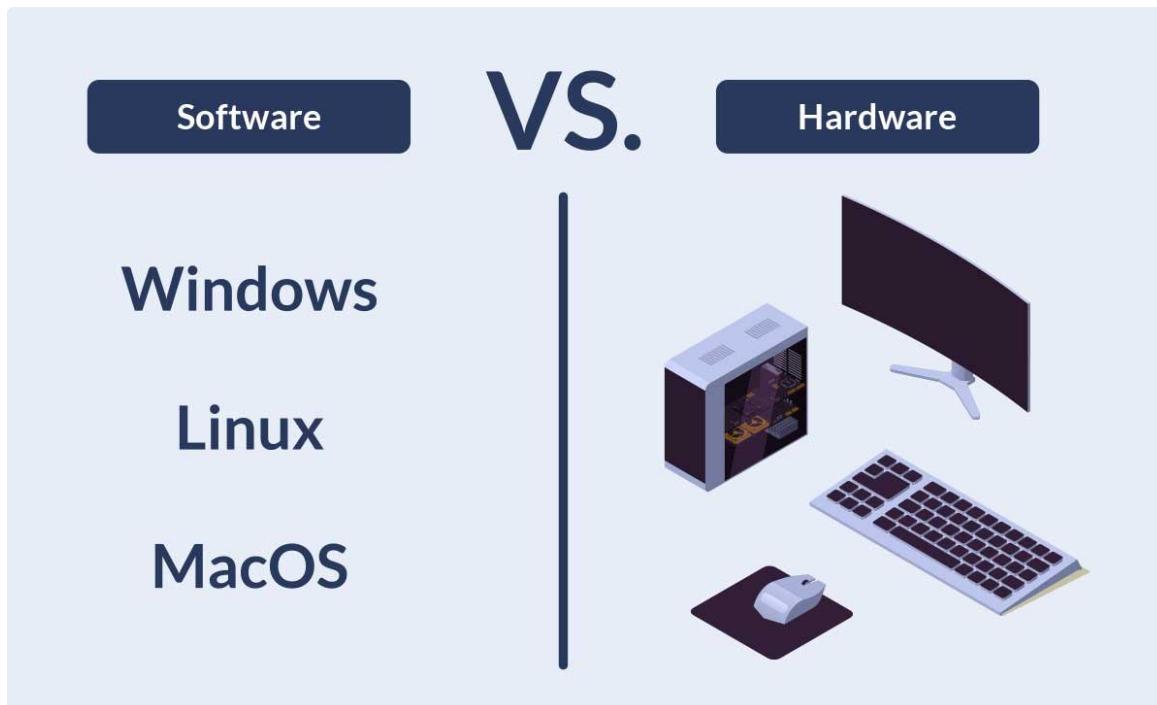
¿Qué son el **hardware** y el **software**?

El **hardware** lo componen las partes físicas del computador, es decir, todo lo que es tangible en él, lo que se puede tocar. Sin embargo, dichas partes por sí solas no hacen que la máquina funcione, requieren de su contraparte intangible: el **software**.

Por su lado, el **software** consiste en un conjunto de datos e instrucciones que le dictan al computador cómo debe trabajar y qué tareas realizar. Estas instrucciones, o programas, son alojadas en un dispositivo electrónico de almacenamiento permanente y ejecutadas por el procesador.

En la *Figura 3* se muestran algunos ejemplos de **hardware** y **software** en un computador.

Figura 3. Hardware vs. Software



¿Cómo trabaja el computador?

Presta atención a la *Figura 4* y observa la forma en la que trabaja un computador y cómo interactúan sus partes.

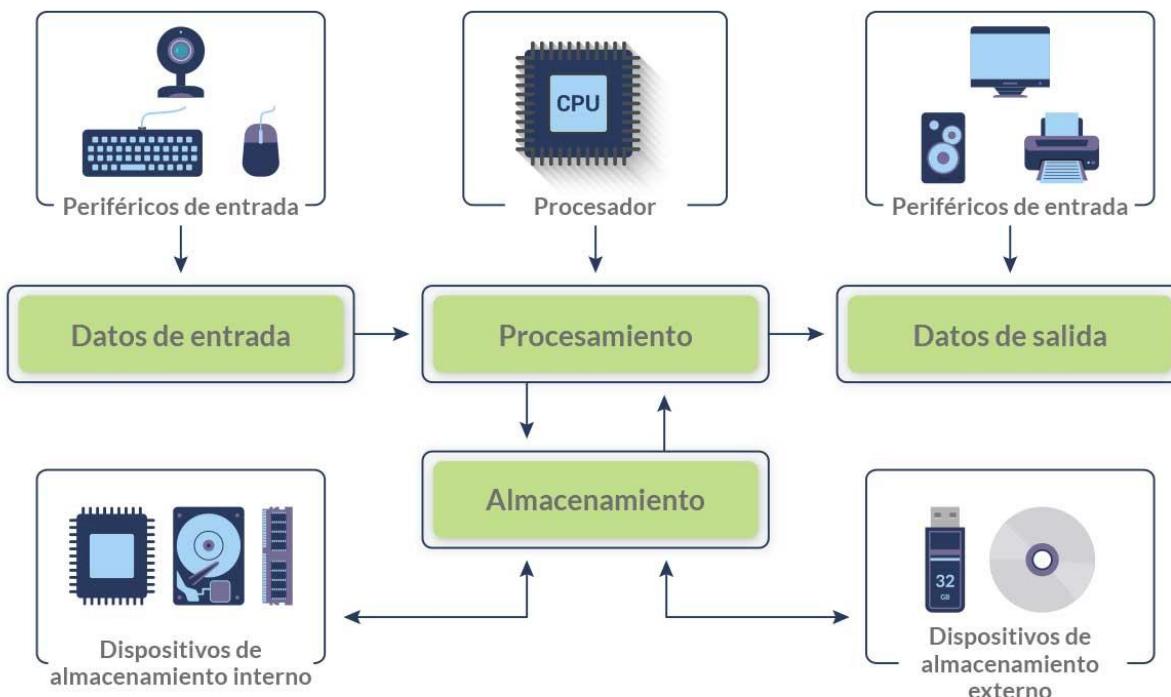
Los datos ingresan al computador a través de periféricos de entrada como el *mouse*, el teclado, la cámara web o el micrófono. Esta información es adaptada a un lenguaje de máquina para que el cerebro del computador, conocido como procesador, pueda realizar operaciones con ella.

Además de procesar la información, el procesador también puede guardarla mediante dispositivos de almacenamiento interno o externo. En el primer grupo encontramos las memorias temporales, las permanentes y los discos duros mecánicos o de estado sólido.

Mientras que, en el segundo, tenemos, entre otros, las memorias *flash* o USB, las memorias SD y los discos ópticos como los DVD.

Una vez procesada la información, se emplean los periféricos de salida para extraerla, como los monitores, los altavoces o las impresoras.

Figura 4. ¿Cómo trabaja el computador?



¿Qué son los sistemas de numeración?

Estos formatos tienen la capacidad de representar cualquier magnitud numérica, cumpliendo ciertas reglas que se establecen de acuerdo con su origen. A manera de ejemplo, existen múltiples sistemas o códigos numéricos distintivos de culturas antiguas, comunidades étnicas o regiones específicas.

Los sistemas de numeración pueden ser posicionales o no posicionales.

En los no posicionales, los símbolos que componen la magnitud no tienen un peso asociado a su posición. En la *Figura 5* se observan dos ejemplos de estos sistemas utilizados en el pasado en Egipto y Roma.

Figura 5. Sistemas de numeración antiguos no posicionales

Sistema de numeración Egipcio		Sistema de numeración Romano
1=	/ staff	I
10=	U heel bone	II
100=	O coil of rope	III
1000=	L lotus flower	IV
10,000=	P pointing finger	V
100,000=	T tadpole	VI
1,000,000=	H astonished man	VII
		VIII
		IX
		X
		XI
		XII

Por su parte, en los sistemas de numeración posicionales, utilizados a menudo en la computación, cada símbolo contiene la magnitud numérica y un peso diferente de acuerdo con su posición. Además, estos sistemas poseen una base y una cantidad determinada de símbolos permitidos.

Para interpretar cada número, es importante saber tres cosas: primero, identificar en qué base de numeración está escrito; segundo, conocer todos los símbolos que componen el sistema de numeración; y tercero, identificar el peso de cada símbolo dentro de la cifra, de acuerdo con su ubicación.

En la *Figura 6* se analiza un número binario, allí se puede observar la base del sistema, que corresponde al subíndice que se encuentra al final del número. En este caso, la base es 2, lo que significa que está escrito en binario.

Para el sistema de numeración binario existen dos símbolos permitidos: el 0 y el 1. Este sistema es posicional, es decir, la ubicación de cada símbolo dentro de la cifra hace que este tenga un peso diferente, de ahí las expresiones $2^0, 2^1, 2^2, \dots 2^n$, donde 2 es la base y n es la potencia que aumenta de acuerdo con la cantidad de cifras, siendo "0" la asociada a la cifra de menor peso, y n a la de mayor.

Para comprender de mejor manera las características de los sistemas de numeración, en las *Figuras 6, 7, 8 y 9* se presentan ejemplos de cada uno de los sistemas que se estudiarán en este curso.

Figura 6. Representación de un número binario

Sistema binario: base 2, símbolos permitidos: 0 y 1

Ejemplo de número binario = 10011_2

The diagram shows the binary number 10011_2 . Above the number, five blue arrows point from left to right, each labeled with a power of 2: 2^4 , 2^3 , 2^2 , 2^1 , and 2^0 .

Peso de la cifra: depende de su ubicación dentro del número, siendo 2^0 la de menor peso, y 2^4 , la de mayor.

Base del número: la base del número es la del subíndice, en este caso, 2, es decir, un número binario.

Símbolos: el número de símbolos es igual al de la base de numeración. En el sistema binario hay dos símbolos: 0 y 1.

Figura 7. Representación de un número octal

Sistema octal: base 8, símbolos permitidos: 0, 1, 2, 3, 4, 5, 6 y 7

Ejemplo de número octal = 407_8

The diagram shows the octal number 407_8 . Above the number, three blue arrows point from left to right, each labeled with a power of 8: 8^2 , 8^1 , and 8^0 .

Peso de la cifra: depende de su ubicación dentro del número, siendo 8^0 la de menor peso, y 8^2 , la de mayor.

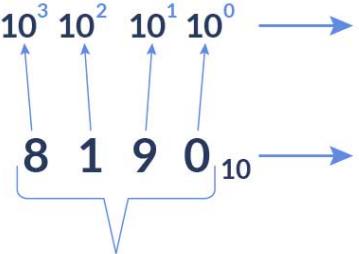
Base del número: la base del número es la del subíndice, en este caso, 8, es decir, un número octal.

Símbolos: el número de símbolos es igual al de la base de numeración. En el sistema octal hay ocho símbolos: 0, 1, 2, 3, 4, 5, 6 y 7.

Figura 8. Representación de un número decimal

Sistema decimal: base **10**, símbolos permitidos:
0, 1, 2, 3, 4, 5, 6, 7, 8 y 9

Ejemplo de número decimal = 8190_{10}



Peso de la cifra: depende de su ubicación dentro del número, siendo 10^0 la de menor peso, y 10^3 , la de mayor.

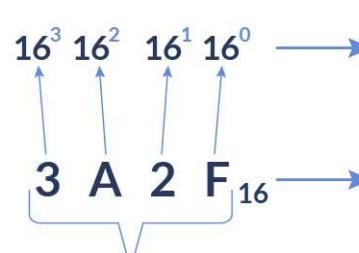
Base del número: la base del número es la del subíndice, en este caso, 10, es decir, un número decimal.

Símbolos: el número de símbolos es igual al de la base de numeración. En el sistema decimal hay 10 símbolos, que son: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

Figura 9. Representación de un número hexadecimal

Sistema hexadecimal: base **16**, símbolos permitidos:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F

Ejemplo de número hexadecimal = $3A2F_{16}$



Peso de la cifra: depende de su ubicación dentro del número, siendo 16^0 la de menor peso, y 16^3 , la de mayor.

Base del número: la base del número es la del subíndice, en este caso, 16, es decir, un número hexadecimal.

Símbolos: el número de símbolos es igual al de la base de numeración. En el sistema hexadecimal hay 16 símbolos, que son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

Conversión entre sistemas de numeración

Conocer la equivalencia entre magnitudes de diferentes sistemas de numeración es de suma importancia, pues durante la programación es común cambiar de formato. Asimismo, es preciso interpretar correctamente la información que ingresa y sale del equipo.

Por ello, a continuación, conoceremos los diferentes procedimientos de conversión, en el siguiente orden:

- 1. Conversiones entre los sistemas de numeración binario, octal, decimal y hexadecimal, al sistema decimal.**
- 2. Conversiones entre el sistema decimal y cada uno de los sistemas de numeración.**
- 3. Conversión entre los sistemas de numeración binario, octal y hexadecimal sin pasar por el sistema decimal.**
- 4. Resumen sobre el procedimiento para realizar las diferentes conversiones.**

¿Cómo se realiza la conversión entre los sistemas binarios, octal y hexadecimal, al sistema decimal?

Para facilitar la comprensión de este procedimiento, las *Figuras 10* y *11* presentan dos ejemplos de dichas conversiones. En la primera figura, se realiza la conversión de binario a decimal, mientras que, en la segunda, se convierte de hexadecimal a decimal.

Por su parte, la *Figura 12* presenta un compendio sobre cómo realizar conversiones entre los sistemas binario, octal, decimal y hexadecimal, al sistema decimal.

Figura 10. Conversión de binario a decimal

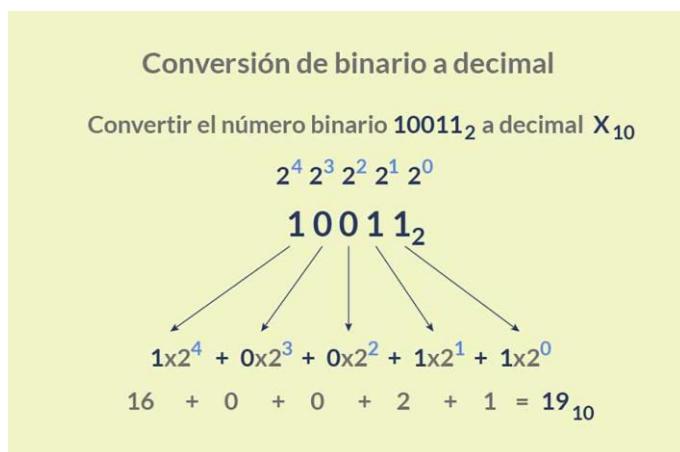
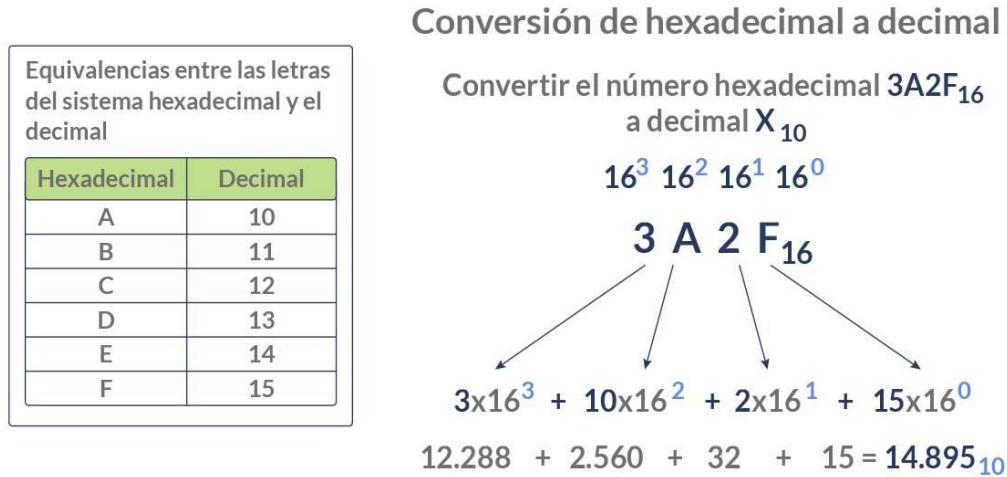


Figura 11. Conversión de hexadecimal a decimal**Figura 12.** Resumen de conversión entre los sistemas binario, octal, decimal y hexadecimal, al sistema decimal

Resumen de conversión entre los sistemas binario, octal, decimal y hexadecimal, al sistema decimal

Nombre	Base (No. de símbolos)	Símbolos	Ejemplos de conversión al sistema decimal
Binario	2	0 y 1	$10011_2 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 16 + 0 + 0 + 2 + 1 = 19_{10}$
Octal	8	0, 1, 2, 3, 4, 5, 6 y 7	$407_8 = 4 \times 8^2 + 0 \times 8^1 + 7 \times 8^0 = 256 + 0 + 7 = 263_{10}$
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8 y 9	$8190_{10} = 8 \times 10^3 + 1 \times 10^2 + 9 \times 10^1 + 0 \times 10^0 = 8.000 + 100 + 90 + 0 = 8190_{10}$
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F	$3A2F_{16} = 3 \times 16^3 + 10 \times 16^2 + 2 \times 16^1 + 15 \times 16^0 = 12.288 + 2.560 + 32 + 15 = 14.895_{10}$

¿Cómo se realiza la conversión entre el sistema decimal, y los sistemas binarios, octal y hexadecimal?

En la *Figura 13* puedes observar la conversión entre un sistema decimal y uno binario. Ahora bien, para esta conversión, debes tener en cuenta el siguiente procedimiento:

- 1.** Realiza divisiones sucesivas entre el número decimal y la base del sistema al cual se pretende convertir, en este caso, el 2.
- 2.** En la parte izquierda, ubicas la parte entera del resultado, mientras que, en la derecha, sitúas el residuo o parte decimal. *Nota: asegúrate de escribir todas las cifras del residuo.*
- 3.** Realiza este procedimiento sucesivamente hasta que la parte entera sea cero (0).
- 4.** Toma cada uno de los residuos con todas sus cifras decimales y multiplícalos por la base, es decir, por 2.
- 5.** Finalmente, escribe el resultado en orden invertido o ascendente, como lo indica la flecha color naranja, es decir, desde abajo hasta arriba:

Figura 13. Ejemplo de conversión del sistema decimal al binario

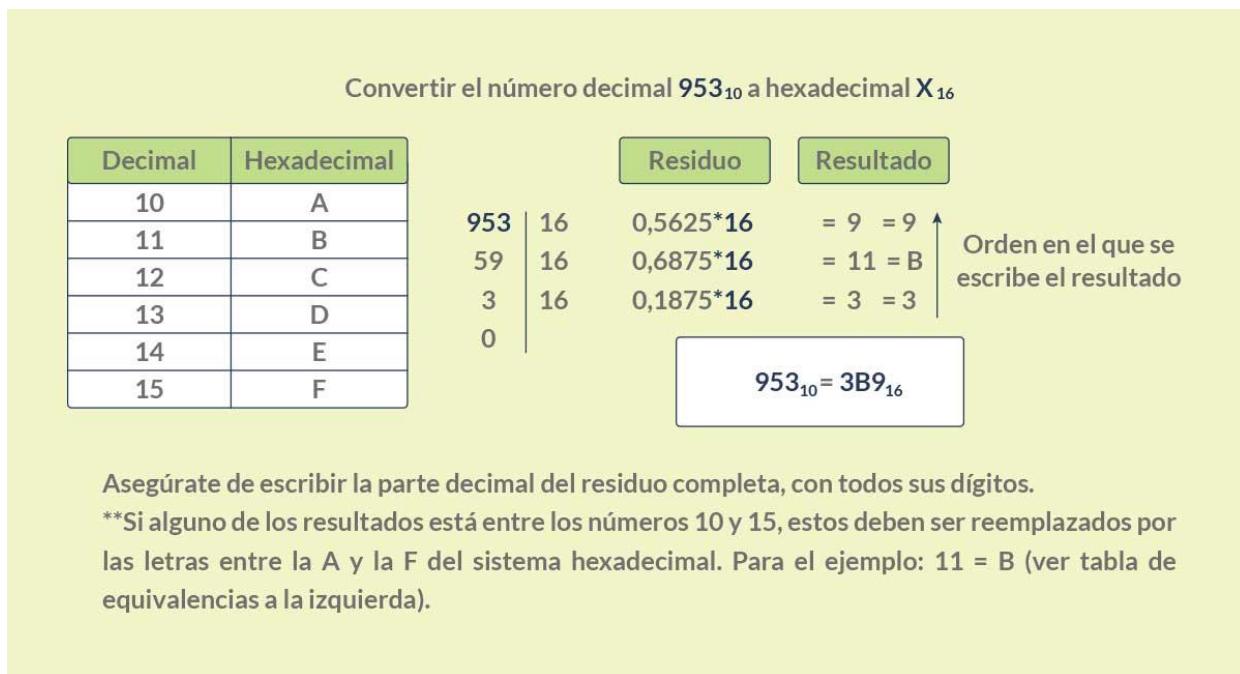
Ejemplo de conversión del sistema decimal al binario			
Convertir el número decimal 953_{10} a binario X_2			
		Residuo	Resultado
953	2	$0,5 \cdot 2$	= 1
476	2	$0 \cdot 2$	= 0
238	2	$0 \cdot 2$	= 0
119	2	$0,5 \cdot 2$	= 1
59	2	$0,5 \cdot 2$	= 1
29	2	$0,5 \cdot 2$	= 1
14	2	0	= 0
7	2	$0,5 \cdot 2$	= 1
3	2	$0,5 \cdot 2$	= 1
1	2	$0,5 \cdot 2$	= 1
0			

Orden en el que se escribe el resultado

$953_{10} = 1110111001_2$

Fíjate ahora en la *Figura 14*, allí se ilustra la conversión entre el sistema decimal y el sistema hexadecimal. En esta ocasión, debes tener especial cuidado al trabajar con todas las cifras del residuo. Además, cuando el resultado del producto entre el residuo y el número 16 sea un número entre el 10 y el 15, este debe ser reemplazado por alguna de las letras equivalentes del sistema hexadecimal, tal como se muestra en la figura.

Figura 14. Ejemplo de conversión del sistema decimal al hexadecimal



¿Cómo se realiza la conversión entre los sistemas binarios, octal y hexadecimal, sin pasar por el sistema decimal?

Para comprender mejor esta conversión, observa con atención la *Figura 15*. Allí se aprecia una tabla de equivalencias, un ejemplo que explica cómo interpretar dicha tabla y, en la parte inferior, el procedimiento para realizar la conversión.

Figura 15. Ejemplo de conversión del sistema binario al octal

Convertir el número binario 10110011_2 a octal X_8

Tabla de conversión			Ejemplo de la tabla		Conversión	
Decimal	Binario	Octal				
0	000	0				
1	001	1				
2	010	2				
3	011	3				
4	100	4				
5	101	5				
6	110	6				
7	111	7				

$$\begin{array}{c} 2^2 \quad 2^1 \quad 2^0 \\ \uparrow \quad \uparrow \quad \uparrow \\ 1 \ 0 \ 1_2 \\ 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \\ 5_{10} = 5_8 \end{array}$$

$$\begin{array}{c} 10110011_2 \\ \underbrace{\hspace{1cm}}_{2} \quad \underbrace{\hspace{1cm}}_{6} \quad \underbrace{\hspace{1cm}}_{3_8} \\ 010110011_2 \end{array}$$

1. Forma grupos de 3 bits, comenzando por el bit menos significativo (LSB).
 2. Si el último grupo no alcanza a tener 3 bits, complétalo con ceros "0".
 3. Reemplaza cada grupo de 3 bits por su equivalente al sistema octal, como muestra la tabla de la izquierda. Observa el ejemplo que permite comprender la tabla.

Igual sucede en la *Figura 16*, que expone un ejemplo de conversión entre un número binario y uno hexadecimal. En ella se presenta una la tabla de equivalencias entre los sistemas decimal, binario y hexadecimal, también un ejemplo sobre cómo interpretar dicha tabla, y el procedimiento para realizar tal conversión.

Figura 16. Ejemplo de conversión del sistema binario al hexadecimal

Convertir el número binario 1111111100_2 a hexadecimal X_{16}

Tabla de conversión			Ejemplo de la tabla		
Decimal	Binario	Hexa	Decimal	Binario	Hexa
0	000	0	8	000	0
1	001	1	9	001	1
2	010	2	10	010	2
3	011	3	11	011	3
4	100	4	12	100	4
5	101	5	13	101	5
6	110	6	14	110	6
7	111	7	15	111	7

$$\begin{array}{c} 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\ 1 \ 1 \ 0 \ 1_2 \\ 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \\ 13_{10} = D_{16} \end{array}$$

$$\begin{array}{c} 1111111100_2 \\ \underbrace{\hspace{1cm}}_{3} \quad \underbrace{\hspace{1cm}}_{F} \quad \underbrace{\hspace{1cm}}_{C}_{16} \\ 00111111100_2 \\ 3 \quad 15 \quad 12_{16} \\ \downarrow \quad \downarrow \quad \downarrow \\ 3 \quad F \quad C_{16} \end{array}$$

1. Forma grupos de 4 bits, comenzando por el bit menos significativo (LSB).
 2. Si el último grupo no alcanza a tener 4 bits, complétalo con ceros "0".
 3. Reemplaza cada grupo de 4 bits por su equivalente al sistema hexadecimal, como muestra la tabla de la izquierda. Observa el ejemplo que permite comprender la tabla.

En la *Figura 17* se presenta un resumen de los procedimientos para realizar las conversiones entre los diferentes sistemas de numeración.

Figura 17. Resumen de procedimientos para realizar conversiones

	BINARIO	OCTAL	DECIMAL	HEXADECIMAL
BINARIO		Formar grupos de 3 bits desde el LSB	Expresión polinómica con base 2 y potencias ascendentes	Formar grupos de 4 bits desde el LSB
OCTAL	Expresar cada dígito con 3 bits		Expresión polinómica con base 8 y potencias ascendentes	Pasar primero a binario y después a hexadecimal
DECIMAL	Divisiones sucesivas entre 2 y residuos *2 (escribir invertido)	Divisiones sucesivas entre 8 y residuos *8 (escribir invertido)		Divisiones sucesivas entre 16 y residuos *16 (escribir invertido)
HEXADECIMAL	Expresar cada símbolo con 4 bits	Pasar primero a binario y después a octal	Expresión polinómica con base 16 y potencias ascendentes	

¿Qué es un bit?

Un bit es la unidad básica o mínima de almacenamiento de información en un computador, por lo que no se puede almacenar menos de esta cantidad.

Por otro lado, solo puede tener dos estados (0 y 1) que representan los niveles lógicos de voltaje que entiende el lenguaje de máquina de las computadoras.

Lee con atención la información de la *Figura 18*, allí se amplía el concepto de bit y cómo se interpreta.

Figura 18. El bit

Bit = Binary Digit o dígito binario

Un bit es la unidad mínima de almacenamiento, solo tiene dos estados (**0 y 1**), lo cual se asemeja a tener un interruptor abierto o cerrado, o una lámpara que está encendida o apagada.



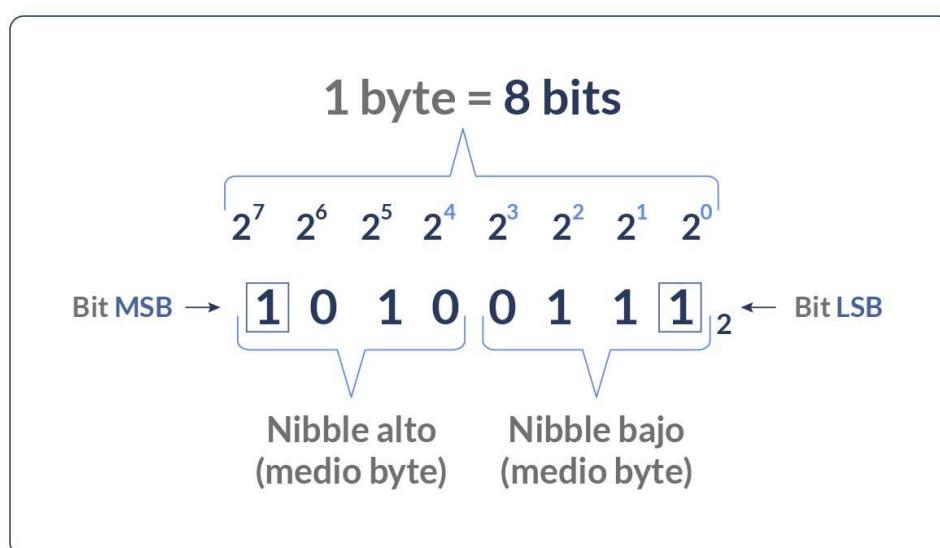
LSB: Least Significant Bit o bit menos significativo (siempre será 2^0)

MSB: Most Significant Bit o bit más significativo (depende de la cantidad de bits)

¿Qué es un byte?

Un byte es un conjunto de 8 bits, es la mínima unidad direccionable por un procesador, además, es la medida estándar de uso frecuente para referirse al almacenamiento de información en computadores y memorias. En la *Figura 19* se ilustra cómo se conforma un byte.

Figura 19. El byte



1.3. Lenguajes de programación

Cuando se habla de programación, no solo en computación, usualmente se hace referencia a la manera de resolver un problema particular. No obstante, para llegar a la solución, se debe considerar cuidosamente el planteamiento, así como interpretar correctamente los requerimientos, los insumos de entrada, los resultados esperados, los recursos de los que se dispone y aquellos que demanda el problema.

Así pues, una vez identificado y comprendido el problema, se procede a esbozar una solución lógica que contenga todos los elementos necesarios para resolver el inconveniente.

Ahora bien, si nos ubicamos en el campo de la computación, después de realizar todo lo anterior, se procede a llevar dicha solución a un código de programación asociado a un lenguaje en particular.

Para aprender más



Si deseas comprender la importancia de aprender a programar en la actualidad, te invitamos a leer el siguiente material:

Columna: Conozca cuáles son los beneficios de aprender a programar
Autor: Tecnósfera El Tiempo

URL: <https://www.eltiempo.com/tecnosfera/novedades-tecnologia/beneficios-de-aprender-a-programar-40497>

En el recurso anterior se plantean varios beneficios que enmarcan la importancia que tiene la programación en la actualidad, en especial, cuando sistemas como computadores, celulares y diversos sistemas electrónicos requieren de procesos algorítmicos y de codificación para su funcionamiento.

¿Qué es un lenguaje de programación?

Un lenguaje de programación es un lenguaje artificial creado para que el hombre pueda controlar una máquina o un computador. En esencia, posibilita la traducción de una solución lógica de un problema determinado a un lenguaje que las máquinas puedan comprender.

Los lenguajes de programación se caracterizan por tener reglas específicas en su estructura, semántica y sintaxis, las cuales el programador debe respetar. Asimismo, existen múltiples lenguajes de programación, cada uno con características únicas más adecuadas para unas aplicaciones que para otras.

¿Por qué Python?

Para efectos de este curso, usaremos el lenguaje de programación Python, debido a su carácter intuitivo y amigable para las personas que están comenzando en la programación. Entre sus ventajas, se destaca una sintaxis simple, una estructura fácil de comprender, seguridad, eficiencia y versatilidad para aplicaciones muy variadas, ya sea en fase de desarrollo o en diferentes tipos de escala, como Raspberry Pi, inteligencia y visión artificial, robótica, *machine learning*, etc.

Otra ventaja de Python, es la facilidad de encontrar una amplia cantidad de librerías, códigos y documentación en las redes sociales, libros y páginas especializadas dedicadas a este lenguaje, así como numerosas comunidades y foros que facilitarán tu avance como programador.

A continuación, definiremos dos términos de uso frecuente en Python para brindarte un mejor contexto.

Script: se denomina así al código fuente escrito, y generalmente interpretado, en un lenguaje de programación determinado, en este caso, Python. Este código debe cumplir con las reglas del lenguaje, es decir, su estructura, sintaxis y semántica. Un *script* contiene operadores, datos y estructuras lógicas que se ejecutan a través de una herramienta llamada *intérprete* para llevar a cabo tareas específicas al interior de un computador.

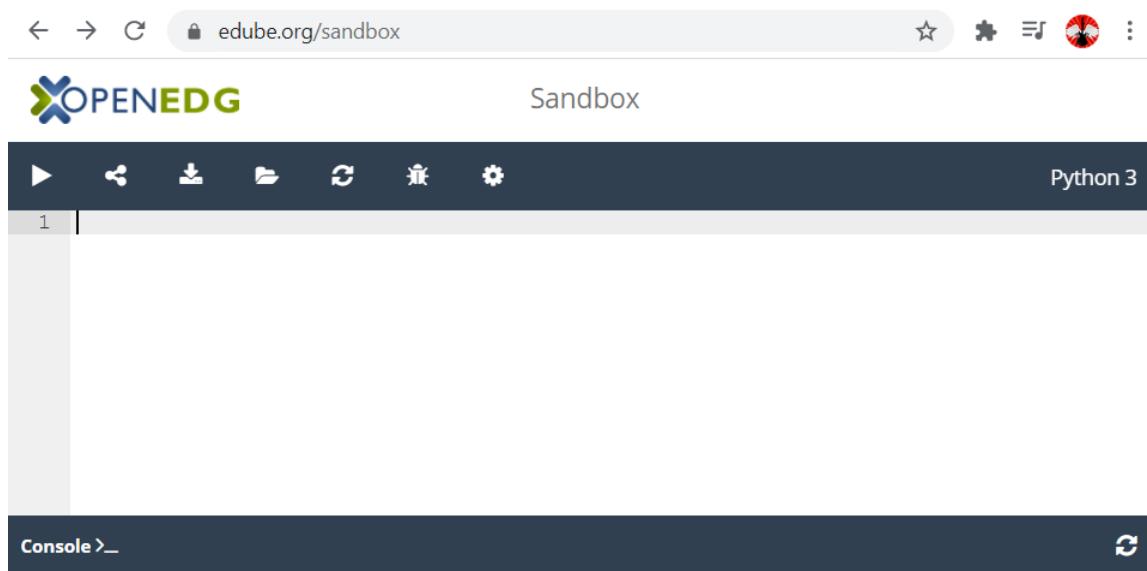
Sandbox o caja de arena: es una herramienta virtual limitada que permite ejecutar el *script* de un lenguaje de programación. Es muy útil, fácil de usar y accesible, por ejemplo, en el caso de Python, puedes encontrar varias opciones gratuitas si escribes en motores de búsqueda “**Sandbox para Python**”. Ten presente que esta será la herramienta que utilizarás en este curso.

Una vez se cuenta con la *Sandbox*, se procede a escribir el código. Para este primer acercamiento, veremos cómo realizar un programa en el que se puedan probar los diferentes tipos de operadores que existen.

En la *Figura 20*, se presenta el entorno de lo que sería un *Sandbox* para Python. En la parte de arriba, donde se lee *Python 3*, se escribe el *script* o código, que será interpretado por el *Sandbox* luego de hacer clic en el botón **Play o Running**. Así pues, si el *script* no presenta errores de sintaxis, podrá ser interpretado.

En la parte de abajo, en *Console*, se obtienen los resultados del código, siempre y cuando, dentro de las instrucciones, exista la visualización por consola o la instrucción “**print ()**”, de otra forma, el *script* será interpretado internamente y los resultados quedarán guardados en el equipo.

Figura 20. Entorno de un Sandbox para Python



Si deseas profundizar acerca de la programación en Python, te recomendamos observar el siguiente video:

Video



- **Autor:** Curso Python para Principiantes
- **Título:** Fazt
- **URL:** <https://www.youtube.com/watch?v=chPhlsHoEPo>

En el recurso anterior observaste la importancia que tiene el aprendizaje del lenguaje de programación dinámico Python en la actualidad, el cual te permitirá realizar codificación elemental o compleja, dependiendo de la problemática que deseas solucionar, así mismo te permitirá desarrollar código de una manera simple, podrás crear diferentes tipos de aplicaciones tanto de escritorio, aplicaciones web, trabajar con inteligencia artificial entre otras aplicaciones, El anterior video también te permitió profundizar en las bases del lenguaje y como iniciar a usarlo de una manera fácil y rápida.

2. Operadores, datos y estructuras de control de flujo

Para programar, es indispensable comprender los diferentes tipos de datos que existen, los operadores que se pueden usar para procesar los datos y las estructuras que controlan el flujo de la información. Lo anterior, porque cada operador, dato y estructura, cambia de acuerdo con el lenguaje de programación que se vaya a emplear.

2.1. Operadores y expresiones aritméticas, relaciones y lógicas

Operadores en Python

Los operadores definen el tipo de operación que se puede realizar con los datos. Existen operadores matemáticos, lógicos y relacionales, cada uno con su propia forma de escritura, también conocida como sintaxis.

- Los operadores matemáticos, como su nombre lo indica, realizan operaciones matemáticas con los datos, por ejemplo: suma, resta, multiplicación, división y módulo.
- Los operadores lógicos, por su parte, permiten realizar operaciones lógicas con los datos, utilizando: AND, OR y XOR.
- Por último, los operadores relacionales establecen comparaciones entre los datos, por ejemplo: *mayor que*, *menor que*, *igual que*, etc.

A continuación, se relacionan los operadores usados en Python.

Operadores matemáticos en Python

Presta atención a la información incluida en las siguientes figuras.

En la *Figura 21*, se muestran los operadores matemáticos, su sintaxis y un ejemplo de cada uno. En la *Figura 22*, se presenta el primer *script* escrito en un *Sandbox*, el cual contiene

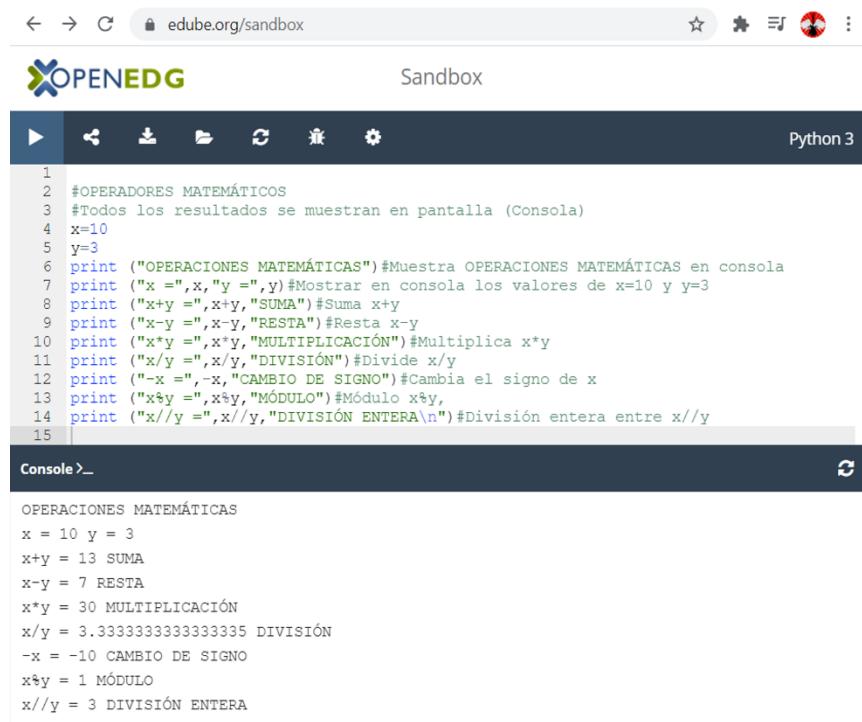
los mismos ejemplos de la figura anterior, pero presentados para cada uno de estos operadores matemáticos.

Recuerda que, en la parte superior del *Sandbox*, se escribe el *script* o código, mientras que, en la parte inferior o consola, aparecen los resultados que se hayan solicitado visualizar desde el *script*.

Figura 21. Operadores matemáticos en Python

TIPO DE OPERADOR	SINTAXIS DEL OPERADOR	EJEMPLO (x=10 , y=3)	RESULTADO
MATEMÁTICO	Suma (+)	x + y	10 + 3 = 13
	Resta (-)	x - y	10 - 3 = 13
	Multiplicación (*)	x * y	10 * 3 = 13
	División (/)	x / y	10 / 3 = 3.3333
	Cambio de signo (multiplicar * -1)	-x	-1*(10) = -10
	Módulo (%)	x % y	10 % 3 = 1
	Exponente (**)	x ** y	10 ** 3 = 1000
	División entera (//)	x // y	10 // 3 = 3

Figura 22. Script escrito en Python con ejemplos de operadores matemáticos



The screenshot shows a Python 3 script running in a browser-based sandbox environment. The script defines variables x=10 and y=3, then performs various operations and prints their results. A large blue callout box on the right side of the interface highlights the execution of instructions and the visualization of results.

```

1 #OPERADORES MATEMÁTICOS
2 #Todos los resultados se muestran en pantalla (Consola)
3
4 x=10
5 y=3
6 print ("OPERACIONES MATEMÁTICAS")#Muestra OPERACIONES MATEMÁTICAS en consola
7 print ("x =",x,"y =",y)#Mostrar en consola los valores de x=10 y y=3
8 print ("x+y =",x+y,"SUMA")#Suma x+y
9 print ("x-y =",x-y,"RESTA")#Resta x-y
10 print ("x*y =",x*y,"MULTIPLICACIÓN")#Multiplica x*y
11 print ("x/y =",x/y,"DIVISIÓN")#Divide x/y
12 print ("-x =",-x,"CAMBIO DE SIGNO")#Cambia el signo de x
13 print ("x%y =",x%y,"MÓDULO")#Módulo x%y,
14 print ("x//y =",x//y,"DIVISIÓN ENTERA\n")#División entera entre x//y
15

```

Console >_

```

OPERACIONES MATEMÁTICAS
x = 10 y = 3
x+y = 13 SUMA
x-y = 7 RESTA
x*y = 30 MULTIPLICACIÓN
x/y = 3.333333333333335 DIVISIÓN
-x = -10 CAMBIO DE SIGNO
x%y = 1 MÓDULO
x//y = 3 DIVISIÓN ENTERA

```

SCRIPT O CÓDIGO QUE EJECUTA INSTRUCCIONES CON OPERADORES MATEMÁTICOS

VISUALIZACIÓN DE RESULTADOS POR CONSOLA

Conozcamos más de cerca un par de operadores matemáticos presentes en el *script*, módulo (%) y división entera (//), puesto que los demás podemos considerarlos de fácil interpretación.

El operador **módulo** entrega como resultado el residuo de una división. Por ejemplo, si tenemos $a=10$ y $b=5$, y se realiza la división entre ellos, es decir, $10/5$, el resultado de la división será 2, porque el 5 cabe exactamente dos veces en el 10, pero no sobra residuo, en consecuencia, el módulo $10\%5 = 0$.

Otro ejemplo, si tenemos $a=7$ y $b=4$, y se realiza la operación módulo $7\%4$, el residuo o módulo sería 3, puesto que el 4 cabe una vez en el 7 y el residuo o lo que sobra, sería 3.

Por su parte, el operador **división entera** realiza divisiones entre dos datos y entrega como resultado solo el componente entero. Por ejemplo, si tenemos $a=7$ y $b=2$, $a//b= 3$, puesto que la parte decimal (0,5) no es tenida en cuenta en el resultado.

Actividad



Se le propone al estudiante replicar el script sobre operadores matemáticos, una vez haya interactuado con la herramienta Sandbox y se haya familiarizado con la sintaxis del lenguaje, le sugerimos realizar operaciones matemáticas mixtas, donde se combinen varias de ellas, también le recomendamos aumentar el nivel de complejidad en las operaciones, y, finalmente, habrá logrado las competencias básicas para el uso de los operadores matemáticos, su sintaxis y dominio de la herramienta Sandbox. En la Figura 23 se presenta una actividad a realizar donde se proponen ejercicios mixtos que combinan diferentes operaciones matemáticas.

Figura 23. Ejercicios propuestos con operadores matemáticos mixtos en Python**EJERCICIOS CON OPERADORES MATEMÁTICOS EN PYTHON**

Siendo $a=20$, $b=4$ y $c=2$, realizar los siguientes ejercicios con operaciones matemáticas usando el Sandbox y visualizando el resultado por consola:

$$\begin{aligned} & a + b - c \\ & a * b - c \\ & b^c + \frac{a}{b} \\ & a \% b - c \\ & b / c * c^5 \\ & 3 * \sqrt[3]{16} \\ & \frac{4 * \sqrt[3]{8}}{b} \end{aligned}$$

Operadores lógicos en Python

Tal como sucedió con los operadores anteriores, presta atención a la información incluida en las siguientes figuras.

En la *Figura 24*, se muestran los operadores lógicos, su sintaxis y un ejemplo de cada uno. Luego, en la *Figura 25*, se presenta el *script* en *Sandbox*, que contiene los mismos ejemplos de la figura anterior, aunque presentados para cada uno de estos operadores lógicos.

Recuerda que, en la parte superior del *Sandbox*, se escribe el *script* o código, mientras que, en la parte inferior o consola, se obtienen los resultados que se solicitaron visualizar desde el *script*.

Figura 24. Operadores lógicos en Python

TIPO DE OPERADOR	SINTAXIS DEL OPERADOR	EJEMPLO (x=10 , y=3)	RESULTADO
LÓGICO	AND (&)	x & y	10 & 3 = 2
	OR ()	x y	10 3 = 11
	XOR (^)	x ^ y	10 ^ 3 = 9
	Complemento A1 (~)	~x	~10 = - 11
	Rotar a la izquierda (<<)	x << y	10 << 3 = 80
	Rotar a la derecha (>>)	x >> y	10 >> 3 = 1

Figura 25. Script escrito en Python con ejemplos de operadores lógicos


The screenshot shows a Python 3 script running in a sandbox environment. The code prints various logical operations between variables x=10 and y=3. The output in the console shows the results of AND, OR, XOR, NOT, left shift, and right shift operations.

```

1 #OPERADORES LÓGICOS
2 #Todos los resultados se muestran en pantalla (Consola)
3 x=10
4 y=3
5 print ("OPERACIONES LÓGICAS") #Muestra OPERACIONES LÓGICAS en consola
6 print ("x =",x,"y =",y) #Mostrar en consola los valores de x=10 y y=3
7 print ("x&y =",x&y,"AND") #AND entre x y y
8 print ("x|y =",x|y,"OR") #OR entre x y y
9 print ("x^y =",x^y,"XOR") #XOR entre x y y
10 print ("~x =",~x,"COMPLEMENTO") #Complemento de x
11 print ("x<<y =",x<<y,"ROTAR A LA IZQUIERDA") #Rotar a x hacia la izquierda tantas veces
12 print ("x>>y =",x>>y,"ROTAR A LA DERECHA\n") #Rotar a x hacia la derecha tantas veces
13
14
15

```

```

OPERACIONES LÓGICAS
x = 10 y = 3
x&y = 2 AND
x|y = 11 OR
x^y = 9 XOR
~x = -11 COMPLEMENTO
x<<y = 80 ROTAR A LA IZQUIERDA
x>>y = 1 ROTAR A LA DERECHA

```

Este tipo de operaciones lógicas requiere la comprensión de funciones básicas, además, para realizarlas, es importante tener en cuenta que estas se interpretan mejor en formato binario, es decir, bit a bit entre los dos datos, coincidiendo los pesos de ambos.

A continuación, en la *Figura 26*, se explican las funciones lógicas más importantes y su tabla de verdad:

Figura 26. Funciones lógicas y sus tablas de verdad

FUNCIÓN AND ($A_0 \& B_0$)			FUNCIÓN OR ($A_0 B_0$)			FUNCIÓN XOR ($A_0 \wedge B_0$)			FUNCIÓN COMPLEMENTO DE A_0 ($\sim A_0$)	
B_0	A_0	SALIDA	B_0	A_0	SALIDA	B_0	A_0	SALIDA	A_0	$\sim A_0$
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1	0	1
1	1	1	1	1	1	1	1	0	1	0

Consejos para cada una de las funciones lógicas

AND -> Esta función solo entregará un alto “1” si los dos bits de entrada están en “1”, si al menos uno de los bits es “0”, la salida será “0”.

OR -> Esta función solo entregará un bajo “0” si los dos bits de entrada están en “0”, si al menos uno de los bits es “1”, la salida será “1”.

XOR -> Esta función entregará un “0” siempre que los dos bits tengan el mismo nivel lógico, si son diferentes, entregará un “1”. Se conoce como una OR exclusiva.

COMPLEMENTO -> Esta función invierte el estado lógico de un bit, es decir, si un bit está en “0”, su complemento será “1” y viceversa. Se conoce como NOT o inversora.

Cuando se realiza una operación lógica entre dos datos, por ejemplo, el dato A y el dato B, realmente se hace entre las parejas de bits que componen los dos datos. Para comprender un poco más este concepto, la Figura 27 ilustra los casos que se realizaron en el script de operadores lógicos. Recuerda que, para comprender con facilidad las operaciones lógicas entre datos, es importante usar las tablas de verdad que hemos explorado anteriormente.

Veamos pues la Figura 27, que contiene los ejercicios lógicos realizados en el script, tomando dos datos para realizar todas las operaciones: X=10 y Y=3. Antes de comenzar, es importante entender que, tanto el 10 como el 3, tienen un equivalente binario, por lo que se debe hacer la conversión para ejecutar y comprender la operación lógica entre ellos. Asimismo, ten en cuenta que cada dato, sea X o Y, aunque estén escritos en decimal, equivalen a un conjunto de 8 bits, es decir, un byte.

El dato X se compone de los bits X₇ X₆ X₅ X₄ X₃ X₂ X₁ X₀, mientras que el Y se compone de los bits Y₇ Y₆ Y₅ Y₄ Y₃ Y₂ Y₁ Y₀. En la primera tabla de la figura, se visualizan los dos datos (X y Y), y su equivalente en binario.

Las operaciones lógicas entre dos datos, se realizan, entonces, entre parejas de bits del mismo peso, por ejemplo: (X₀,Y₀), (X₁,Y₁), y así sucesivamente. El resultado de la operación se lee como un dato completo de 8 bits, como se muestra en las tres tablas restantes de cada función, y que se puede cotejar con los resultados por consola del script.

Figura 27. Operadores lógicos AND, OR y XOR, según script

DATO X ₁₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Bits	X ₇	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀
X = 10 ₁₀	0	0	0	0	1	0	1	0
Bits	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
Y = 3 ₁₀	0	0	0	0	0	0	1	1

AND									OR									XOR								
DATO	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	DATO	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	DATO	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
X = 10	0	0	0	0	1	0	1	0	X = 10	0	0	0	0	1	0	1	0	X = 10	0	0	0	0	1	0	1	0
Y = 3	0	0	0	0	0	0	1	1	Y = 3	0	0	0	0	0	0	1	1	Y = 3	0	0	0	0	0	0	1	1
AND	0	0	0	0	0	0	1	0	OR	0	0	0	0	1	0	1	1	XOR	0	0	0	0	0	1	0	1
X & Y = 2 ₁₀	0	0	0	0	0	0	1	0	X Y = 11 ₁₀	0	0	0	0	1	0	1	1	X ^ Y = 9 ₁₀	0	0	0	0	1	0	0	1

Dos funciones lógicas igualmente usadas con frecuencia son: rotar un dato a la izquierda y rotar a la derecha. Ambas operaciones desplazan un bit en un sentido, tantas veces como se le indique en la instrucción. En el caso del ejercicio, la instrucción fue rotar a la izquierda el dato X, es decir, el número 10, tantas veces como la magnitud de Y, es decir, 3 veces.

Igual que en los ejemplos anteriores, el primer paso es convertir los datos de decimal a binario. En la *Figura 28* se presenta el análisis de las dos instrucciones, allí se explican los resultados parciales de las rotaciones hasta llegar al resultado definitivo una vez se ejecuten las tres rotaciones. Te recomendamos revisar el *script* y compararlo con el análisis de cada instrucción.

Figura 28. Operadores lógicos de rotación a la derecha y a la izquierda según script

INSTRUCCIÓN ROTAR A LA IZQUIERDA

Dato X = 10 (00001010 ₂), Dato Y = 3 (00000011 ₂)								
Operación = x<<y, Rotar x a la izquierda 3 veces (el valor de y)								
2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	D ₇
0	0	0	0	1	0	1	0	D ₇
0	0	0	1	0	1	0	0	D ₆
0	0	1	0	1	0	0	0	D ₅
0	1	0	1	0	0	0	0	D ₄
								D ₃
								D ₂
								D ₁
								D ₀

La rotación a la izquierda equivale a multiplicar un dato por 2.

- Dato x = 10₁₀ = 00001010₂ rotar 3 veces
- Rotación 1 de x = 00010100₂ = 20₁₀
- Rotación 2 de x = 00101000₂ = 40₁₀
- Rotación 3 de x = 01010000₂ = 80₁₀

INSTRUCCIÓN ROTAR A LA DERECHA

Dato X = 10 (00001010 ₂), Dato Y = 3 (00000011 ₂)								
Operación = x>>y, Rotar x a la derecha 3 veces (el valor de y)								
2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	D ₇
0	0	0	0	1	0	1	0	D ₇
0	0	0	0	0	1	0	1	D ₆
0	0	0	0	0	0	1	0	D ₅
0	0	0	0	0	0	0	1	D ₄
								D ₃
								D ₂
								D ₁
								D ₀

La rotación a la derecha equivale a dividir un dato entre 2.

- Dato x = 10₁₀ = 00001010₂ rotar 3 veces
- Rotación 1 de x = 00000101₂ = 5₁₀
- Rotación 2 de x = 00000010₂ = 2₁₀
- Rotación 3 de x = 00000001₂ = 1₁₀



Actividad

Se le propone al estudiante usar la herramienta Sandbox y replicar el script con las instrucciones donde se usan los operadores lógicos, una vez haya interactuado con la herramienta, realizar operaciones lógicas mixtas, donde se combinen varias de ellas, posteriormente, cambiar los ejercicios por unos de mayor complejidad analizando los resultados. El estudiante desarrollará las competencias básicas para el uso de los operadores lógicos, su correcta sintaxis y mayor afianzamiento en el uso de la herramienta Sandbox. En la Figura 29 se presenta una actividad a realizar donde se proponen ejercicios mixtos que combinan diferentes operaciones lógicas.

Figura 29. Ejercicios propuestos con operadores lógicos mixtos en Python

EJERCICIOS CON OPERADORES LÓGICOS EN PYTHON

Siendo $a=5$, $b=15$ y $c=64$, realizar los siguientes ejercicios con operaciones lógicas usando el Sandobox y visualizando el resultado por consola:

$a \& b \& c$
 $a | b | c$
 $a ^ b ^ c$
 $a \& (b ^ c)$
 $b \ll a$
 $(b \ll 2) | c$

Operadores relacionales en Python

Para definir de mejor manera los operadores relacionales, en la *Figura 30* se muestran cuáles son, su sintaxis y un ejemplo de cada uno. Posteriormente, en la *Figura 31*, se presenta un *script* escrito en el *Sandbox*, que contiene los mismos ejemplos para cada uno de estos operadores relacionales.

Los operadores relacionales están hechos para indagar sobre la relación entre dos datos, es decir, se utilizan para saber cómo es un dato frente a otro, por ejemplo, si es *mayor*, *igual*, *diferente*, o *mayor que*.

Debido a que este tipo de operadores es usado para realizar preguntas, entonces, la respuesta esperada será un valor verdadero (*True*) o uno falso (*False*). En el *script* se observa que estos operadores son muy sencillos de interpretar, por lo que los ejemplos usados no requieren mayor análisis.

Figura 30. Operadores relacionales en Python

TIPO DE OPERADOR	SINTAXIS DEL OPERADOR	EJEMPLO (x=10 , y=3)	RESULTADO
RELACIONAL	¿Menor que? (<)	x < y	¿10 < 3? = False
	¿Mayor que? (>)	x > y	¿10 > 3? = True
	¿Menor o igual que? (<=)	x <= y	¿10 <= 3? = False
	¿Mayor o igual que? (>=)	x >= y	¿10 >= 3? = True
	¿Igual? (==)	x = y	¿10 == 3? = False
	¿Diferente? (!=)	x != y	¿10 != 3? = True

Figura 31. Script escrito en Python con ejemplos de operadores relacionales



```

OPENEDG Sandbox
Python 3

1 #OPERADORES RELACIONALES
2 #Todos los resultados se muestran en pantalla (Consola)
3 x=10
4 y=3
5 print ("OPERACIONES RELACIONALES")#Muestra OPERACIONES RELACIONALES en consola
6 print ("x =",x,"y =",y)#Mostrar en consola los valores de x=10 y y=3
7 print ("x<y =",x<y,"¿x MENOR QUE y?")#Pregunta si x es menor que y
8 print ("x>y =",x>y,"¿x MAYOR QUE y?")#Pregunta si x es mayor que y
9 print ("x<=y =",x<=y,"¿x MENOR O IGUAL QUE y?")#Pregunta si x es menor o igual que y
10 print ("x>=y =",x>=y,"¿x MAYOR O IGUAL QUE y?")#Pregunta si x es mayor o igual que y
11 print ("x==y =",x==y,"¿x MAYOR O IGUAL QUE y?")#Pregunta si x es mayor o igual que y
12 print ("x==y",x==y,"¿x IGUAL QUE y?")#Pregunta si x es igual que y
13 print ("x!=y =",x!=y,"¿x ES DIFERENTE DE y?\n")#Pregunta si x es diferente que y
14

Console >_
OPERACIONES RELACIONALES
x = 10 y = 3
x<y = False ¿x MENOR QUE y?
x>y = True ¿x MAYOR QUE y?
x<=y = False ¿x MENOR O IGUAL QUE y?
x>=y = True ¿x MAYOR O IGUAL QUE y?
x==y False ¿x IGUAL QUE y?
x!=y = True ¿x ES DIFERENTE DE y?

```

Actividad



Se propone al estudiante replicar el script con las instrucciones donde se usan los operadores relacionales, una vez haya interactuado con la herramienta, realizar operaciones mixtas, también les sugerimos realizar ejercicios de mayor complejidad y analizar los resultados. El estudiante desarrollará las competencias básicas para el uso de los operadores relacionales, su correcta sintaxis y mayor afianzamiento en el uso de la herramienta Sandbox. En la Figura 32 se presenta una actividad a realizar donde se proponen ejercicios mixtos con este tipo de operadores.

Figura 32. Ejercicios propuestos con operadores relacionales mixtos en Python

EJERCICIOS CON OPERADORES RELACIONALES EN PYTHON

Siendo $a=20$, $b=5$ y $c=5$, realizar los siguientes ejercicios con operaciones relacionales usando el Sandbox y visualizando el resultado por consola:

$$a > b$$

$$a == b$$

$$a == b == c$$

$$a != b$$

$$a \leq b$$

$$b \geq c$$

Tipos de datos en Python

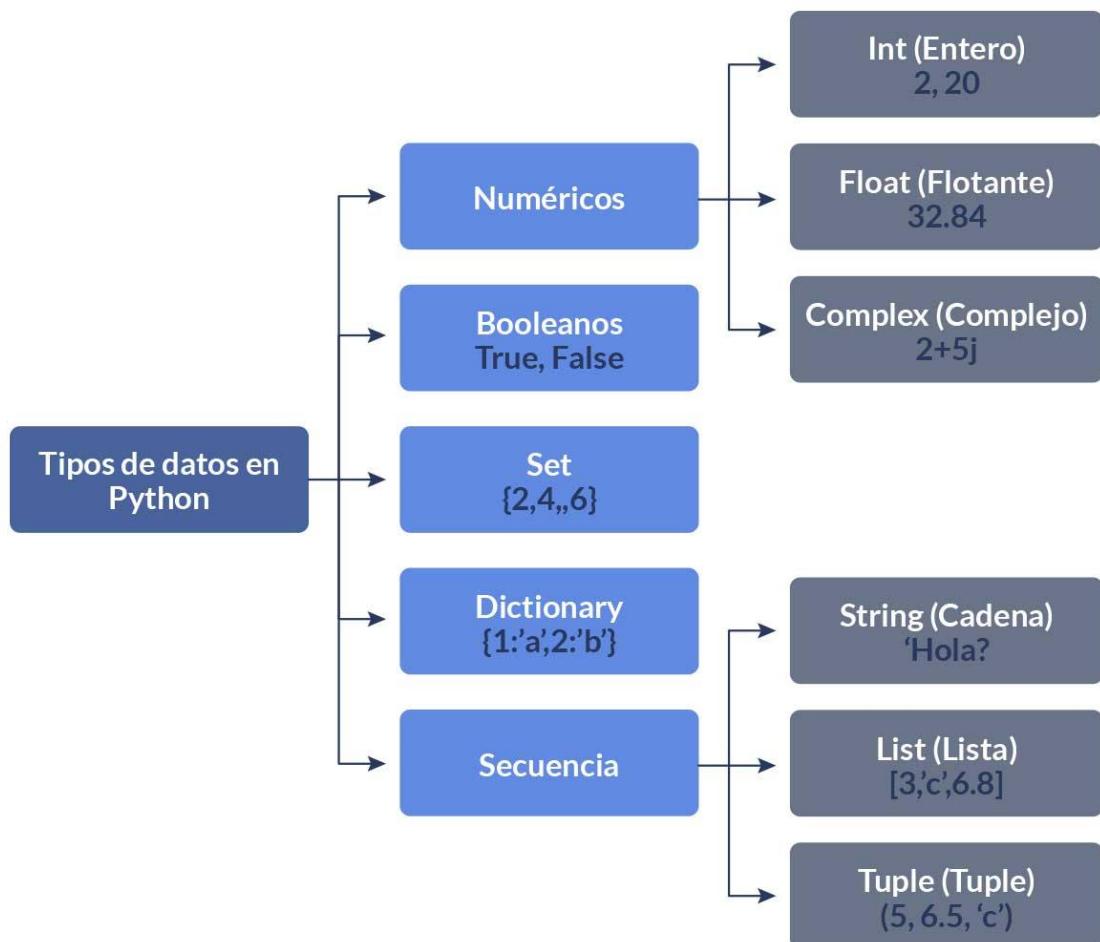
Un dato es información que ocupa un espacio de memoria en el equipo de cómputo, este puede tener diferentes formatos, podría tratarse de un dato numérico, sin embargo, este a su vez tiene múltiples ramificaciones, como es el caso de los números naturales, los enteros, los complejos, los racionales, entre otros. Un dato también podría ser un carácter, como por ejemplo una letra o un símbolo en particular como los del teclado, pero podría ser también una cadena de caracteres.

Elegir el tipo de datos a usar en un programa es de suma importancia, pues esta elección depende de las operaciones que se realizarán con la información y con el tipo de resultado esperado. Por ejemplo, si en un código se requiere un conteo ascendente hasta el número

100, en este caso el tipo de dato recomendado sería un número entero, porque sus características son suficientes para dicha tarea, pero, si por el contrario, el tipo de información que se espera almacenar contiene cifras decimales, como sería el resultado de operaciones con fracciones, raíces, logaritmos, entre otras, en este caso, un dato de tipo entero sería insuficiente, se tendrá que usar un dato de tipo flotante, lo que a su vez se traduce en un dato que consume más recurso de almacenamiento de la máquina, así pues, cada tipo de dato ocupa un espacio de almacenamiento diferente en un equipo. La unidad utilizada para hacer referencia al almacenamiento en un sistema es el Byte.

En la Figura 33 se presenta la forma en la que están clasificados los datos en Python, con algunos ejemplos.

Figura 33. Tipos de datos en Python y algunos ejemplos



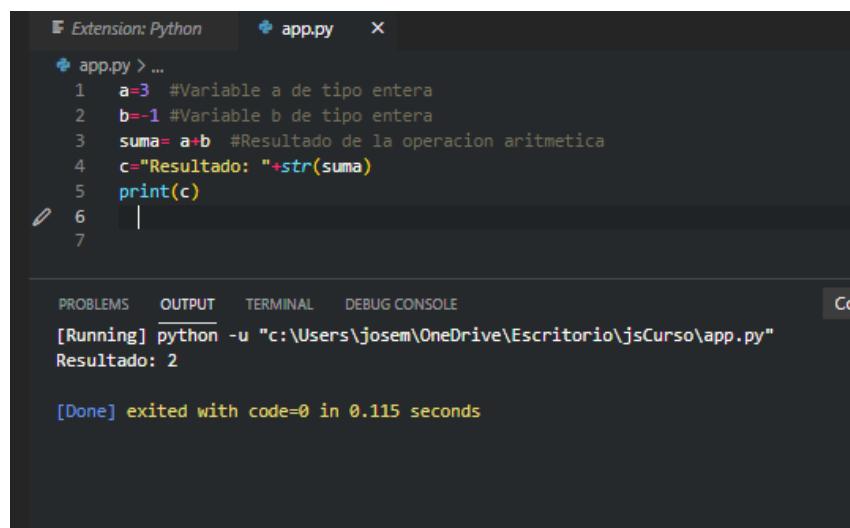
2.2. Datos numéricos, lógicos y de texto

Datos numéricos

Python tiene tres clases de datos de tipo numérico-básicos: números de coma flotante (float, que simulan los números reales), números enteros y números complejos.

1. Números enteros: datos que suelen ser definidos con la notación “int”, los cuales incluyen todos los números enteros. Debido a que se trata de un conjunto de números infinito, los datos de tipo entero están restringidos al espacio de memoria definido por la capacidad de almacenamiento del dispositivo en el que está siendo ejecutado el programa (*Figura 34*).

Figura 34. Ejemplo de números enteros



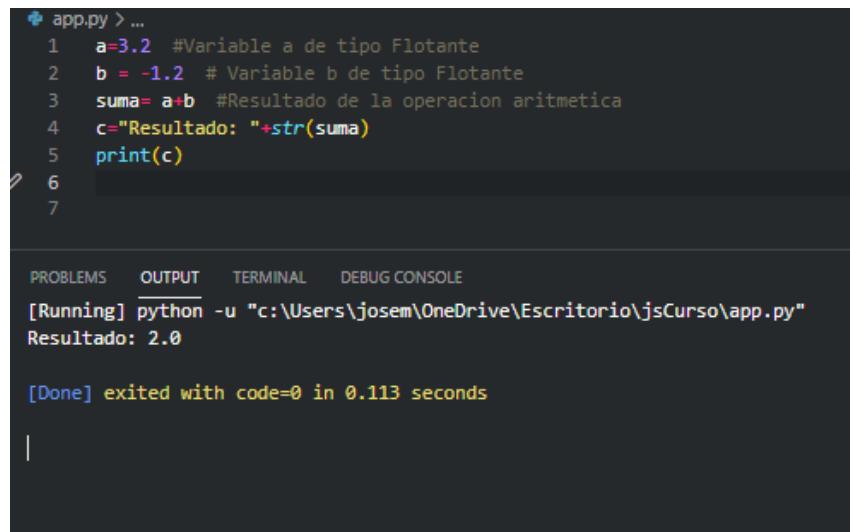
The screenshot shows a code editor window with a Python file named "app.py". The code defines two integers, a=3 and b=-1, performs their multiplication, converts the result to a string, and prints it. The output terminal shows the execution of the script and the resulting output "Resultado: 2".

```
Extension: Python    app.py    X
app.py > ...
1 a=3 #Variable a de tipo entera
2 b=-1 #Variable b de tipo entera
3 suma= a*b #Resultado de la operacion aritmetica
4 c="Resultado: "+str(suma)
5 print(c)
6 |
7

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
Resultado: 2

[Done] exited with code=0 in 0.115 seconds
```

2. Números de coma flotante: aquellos que se usan para representar números reales, aunque también pueden incluir los enteros a manera de conjunto numérico.

Figura 35. Ejemplo de números de coma flotante


```

app.py > ...
1  a=3.2 #Variable a de tipo Flotante
2  b = -1.2 # Variable b de tipo Flotante
3  suma= a+b #Resultado de la operacion aritmetica
4  c="Resultado: "+str(suma)
5  print(c)
6
7

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
Resultado: 2.0

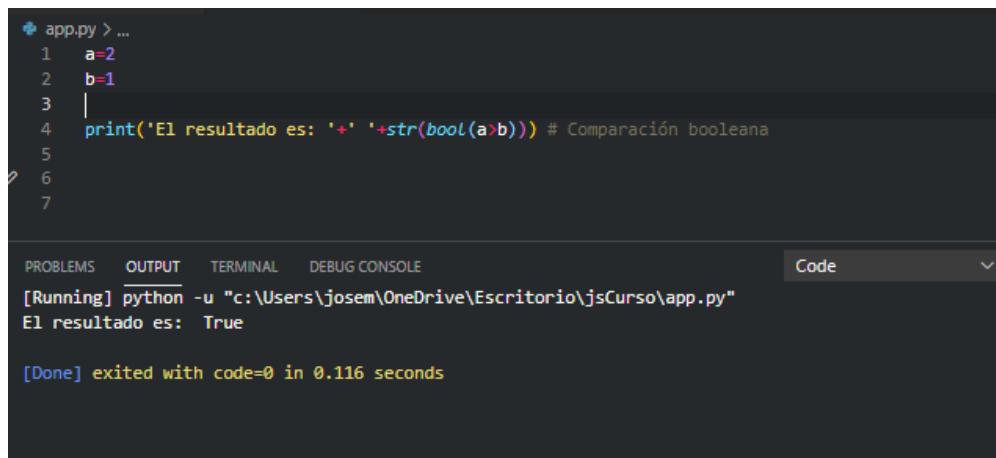
[Done] exited with code=0 in 0.113 seconds
|

```

- 3.** Números complejos: tienen un componente imaginario y uno real, usados generalmente para realizar representaciones algebraicas y análisis matemáticos en distintas áreas de las ciencias e ingenierías.

Datos lógicos

Los datos lógicos o datos boléanos son aquellos que tienen la posibilidad de tener dos valores: 0 o 1, los cuales, generalmente, representan *falso* o *verdadero*, *sí* o *no*. Usualmente, se acompañan de operadores relacionales para generar un dato, como se observa en las *Figuras 36a* y *36b*.

Figura 36a. Ejemplo de comparación booleana


```

app.py > ...
1  a=2
2  b=1
3
4  print('El resultado es: '+ str(bool(a>b))) # Comparación booleana
5
6
7

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
El resultado es: True

[Done] exited with code=0 in 0.116 seconds

```

Figura 36b. Ejemplo de comparación booleana

```
app.py > ...
1 a=1
2 b=2
3
4 print('El resultado es: '+' '+str(bool(a>b))) # Comparación booleana
5
6
7

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Code
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
El resultado es: False

[Done] exited with code=0 in 0.118 seconds
```

Datos de tipo texto

Los datos de tipo texto suelen ser alfanuméricos, comúnmente utilizados para representar e identificar letras del alfabeto latín y números arábigos. Se denominan *Char* si es un solo carácter o *String* cuando se trata de una cadena de caracteres.

Un *Char* suele ser acompañado por dos comillas simples (""), mientras que un *String* de comillas dobles (*Figura 37*).

Figura 37. Ejemplo de dato tipo *Char* y *String*

```

app.py > ...
1 a='j'
2 b="jose"
3
4 print(a+' '+b)
5
6
7

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
j jose

[Done] exited with code=0 in 0.136 seconds
|

```

2.3. Estructuras de control de flujo

Las estructuras de control de flujo son instrucciones que nos permiten evaluar, cualquier cantidad de veces, si es posible seguir un flujo para cumplir o no una condición.

Conozcamos algunas de ellas:

1. **Estructura If-Else:** la estructura de control *if* se utiliza para evaluar si una condición es falsa o verdadera, y con base en el resultado, elegir un respectivo flujo para ejecutar otras sentencias (*Figuras 38a y 38b*).

Figura 38a. Ejemplo de estructura If-Else

```

app.py > ...
1 a=1
2 b=2
3 if a>b:
4     print('A mayor que B')
5 if a<b:
6     print('B mayor que A')
7
8

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
B mayor que A

[Done] exited with code=0 in 0.121 seconds
|

```

Figura 38b. Ejemplo de estructura If-Else

```
app.py > ...
1  a=2
2  b=1
3  if a>b:
4      print('A mayor que B')
5  if a<b:
6      print('B mayor que A')
7
8

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
A mayor que B

[Done] exited with code=0 in 0.109 seconds
```

2. Estructura *For*: la estructura de control *For* es aquella en la que se conoce desde el inicio el número de iteraciones del ciclo, por ende, no requiere un condicional de salida para el bucle. En su lugar, se requiere un contador que defina la cantidad de iteraciones que tendrá dicho ciclo.

Figura 39. Ejemplo del ciclo *For*

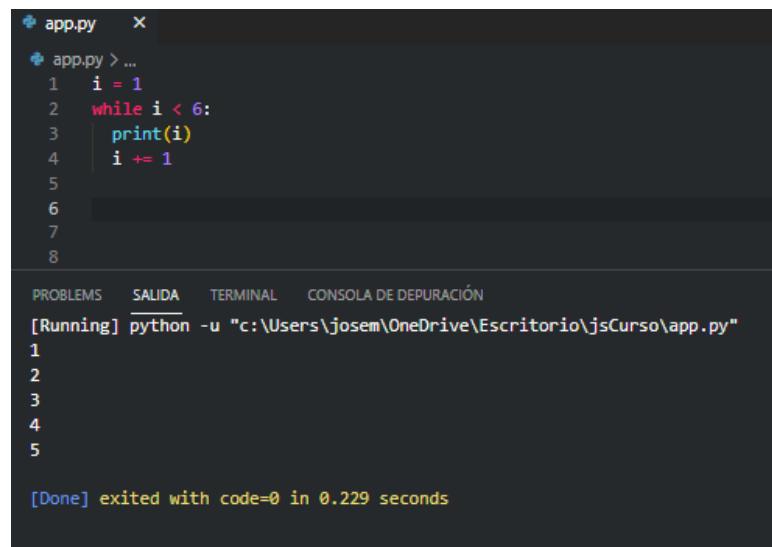
```
app.py > ...
1  for i in range(2):
2      print(i)
3
4
5
6
7
8

PROBLEMS    SALIDA    TERMINAL    CONSOLA DE DEPURACIÓN
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
0
1

[Done] exited with code=0 in 0.199 seconds
```

3. Estructura *While*: la estructura de control *While* es una sentencia de control de flujo que se utiliza para realizar un bloque de sentencias de forma continua mientras se cumple una condición específica.

Figura 40. Ejemplo del ciclo *While*



```

app.py  x
app.py > ...
1  i = 1
2  while i < 6:
3      print(i)
4      i += 1
5
6
7
8

PROBLEMS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
1
2
3
4
5

[Done] exited with code=0 in 0.229 seconds

```

- 4. Estructura *case*:** la estructura de control *switch case* se utiliza cuando se requiere ejecutar diferentes secuencias o flujos con base en el valor de una variable. Es decir, el flujo depende del valor de la variable. Aunque Python no tiene propiamente la estructura *case*, se simulará como se muestra en la *Figura 41*.

Figura 41. Ejemplo de *switch case*



```

app.py > ...
1  condicion=2
2  if condicion == 1:
3      print("Haz a")
4  elif condicion == 2:
5      print("Haz b")
6  elif condicion == 3:
7      print("Haz c")
8  else:
9      print("Haz d")
10 |
11

PROBLEMS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
Haz b

[Done] exited with code=0 in 0.124 seconds

```

En la siguiente página web encontrarás una guía introductoria acerca de las características esenciales de Python. En ella hallarás información sobre las diferentes estructuras de control y algunas generalidades del lenguaje que te serán útiles en este proceso de aprendizaje. Si deseas profundizar en dichos temas, haz clic en el enlace del recuadro.

Para aprender más



Si deseas conocer las generalidades de Python de manera más profunda, te invitamos a leer el siguiente material:

- **Página web:** Tutorial de Python
- **URL:** <https://docs.python.org/es/3/tutorial/>

Para practicar los ejercicios aquí planteados y realizar otros similares, profundizando así en los temas estudiados, te invitamos a explorar el siguiente recurso interactivo:

A green circular icon containing a white laptop computer with two gears on its screen.	Recurso interactivo
	Autor: Codecademy
	Título: Learn Python
	URL: https://www.codecademy.com/learn/learn-python/modules/learn-python-python-syntax-u-6

2.4. Constantes, variables, contadores y acumuladores

Constantes: las constantes son valores que permanecen estáticos a lo largo de todo el programa, es decir, su valor no cambia. Corresponden a un valor de memoria fijo, reservado en la memoria principal de una computadora. Por ejemplo, el valor de Euler, e=2.7182.

Variables: las variables son un espacio de memoria en una computadora destinado a almacenar información específica de un tipo de dato. Es decir, define si va a almacenar caracteres numéricos o valores lógicos. De ser necesario, el valor de la variable puede cambiar en el flujo del programa (*Figura 42*).

Figura 42. Ejemplo de variables

```
app.py > ...
1 a=2
2 print('El dato que tiene la variable es: '+str(a))
3 b=3
4 print('El dato que tiene la variable es: '+str(b))
5 b=4
6 print('El dato que tiene la variable es: '+str(b))

PROBLEMS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
El dato que tiene la variable es: 2
El dato que tiene la variable es: 3
El dato que tiene la variable es: 4

[Done] exited with code=0 in 0.125 seconds
```

Contadores: los contadores son variables de tipo entero que incrementan o decrementan su valor con base a una regla establecida, como se muestra en la *Figura 43*.

Figura 43. Ejemplos de contadores

```
app.py > ...
1 #Se inicia el contador
2 cont=2
3 #Se incrementa el contador
4 cont=cont+1
5 #Tambien se puede usar la siguiente estructura para incrementar el valor
6 #cont +=1
7 print('El valor actual del contador es :'+str(cont))

PROBLEMS SALIDA TERMINAL CONSOLA DE DEPURACIÓN Code
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
El valor actual del contador es :3

[Done] exited with code=0 in 0.114 seconds
```

Acumuladores: los acumuladores son una variable numérica encargada de aglutinar parcialmente operaciones. Asimismo, son una versión ampliada de los contadores y tienen sus mismas características, pero la manera en la que incrementa y decremente el valor de conteo es variable.

Figura 44. Ejemplos de acumuladores

```

app.py > ...
1 #Se inicia el contador
2 cont=2
3 x=2
4 #Se incrementa el contador
5 cont=cont+x
6 print('El valor actual del contardor es :'+str(cont))
7 x = 4
8 cont = cont+x
9 print('El valor actual del contardor es :'+str(cont))
10

PROBLEMS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
[Running] python -u "c:\Users\josem\OneDrive\Escritorio\jsCurso\app.py"
El valor actual del contardor es :4
El valor actual del contardor es :8

[Done] exited with code=0 in 0.12 seconds
|
```

2.5. Notación algorítmica de expresiones aritméticas

La notación algorítmica de expresiones aritméticas es la construcción algebraica de una ecuación, la cual utiliza el dispositivo intérprete (computadora u ordenador) para realizar de manera correcta las operaciones que le proporcionemos. Es decir, para que las ecuaciones que utilizamos cotidianamente se puedan realizar dentro de un algoritmo sin ningún problema (*Figura 45*).

Por ejemplo, veamos en la *Figura 45* cómo convertir la siguiente ecuación a una notación algorítmica:

$$\frac{x}{x+1} + \frac{x}{x+2} = 3$$

Figura 45. Ejemplo notación algorítmica de expresiones aritméticas

```

2      (x/(x+1)) + (x/(x+2)) = 3
3
```

3. Los algoritmos

Un algoritmo es una cadena lógica y finita de pasos a seguir que permite darle solución a un problema cumpliendo ciertos objetivos.

Los algoritmos deben ser eficientes y precisos, y seguir un orden lógico de ejecución para cada uno de los pasos requeridos. El algoritmo debe tener repetibilidad, es decir, si se ejecuta n veces bajo las mismas circunstancias, debe arrojar el mismo resultado. Además, debe ser finito, o sea, tener un estado de inicio y, posteriormente, ofrecer una salida o resultado como solución de un problema específico (Cadavid *et al.*, 2016).

Resolución de problemas con algoritmos

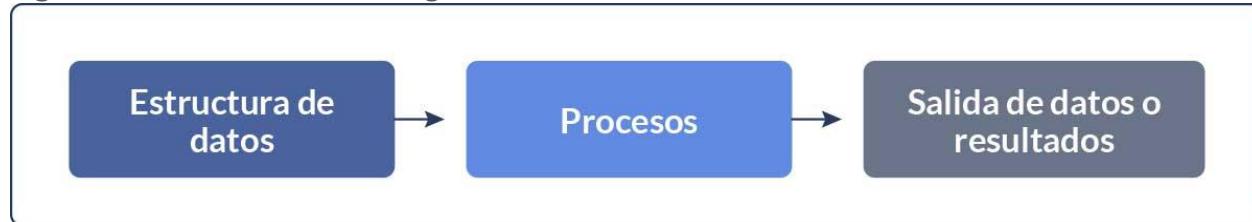
Cuando se planea realizar un algoritmo, es recomendable tener presente los siguientes pasos:

- Tener claridad sobre el problema que se desea solucionar.
- Establecer objetivos que permitan medir la calidad de la solución.
- Diseñar y construir un algoritmo que dé solución al problema.
- Elaborar pruebas al algoritmo que permitan validar su óptimo funcionamiento.

3.1. Estructura de un algoritmo

Los algoritmos comúnmente suelen tener tres secciones principales: entrada de datos, procesos y salida de datos o resultado.

Figura 46. Estructura de un algoritmo



Entrada de datos: es el punto donde se define qué y cuáles datos se ingresarán al algoritmo para ser transformados.

Proceso: es el conjunto de instrucciones y operaciones a ejecutar para darle solución al problema.

Salida: son el conjunto de soluciones y/o resultados que se generan a través de la ejecución de los procesos (Baños & Hernández, 2012).

3.2. Representación: pseudocódigo y diagrama de flujo

Cuando llega el momento de realizar la primera configuración o diseño de un algoritmo, podemos elegir entre los siguientes métodos:

Pseudocódigo

Utilizado para representar una solución algorítmica empleando lenguaje natural escrito y estableciendo los procesos para resolver la problemática de manera clara. Ejemplo:

Proceso

```
Leer Lista_de_variables;  
Ingresar Datos;  
Operar Datos;  
Validar Datos;  
Escribir Expresiones;  
Salida Datos;
```

FinProceso

Diagrama de flujo

Un diagrama de flujo es una representación visual de un algoritmo que nos permite observar de manera gráfica las secuencias a seguir para alcanzar una solución a una problemática particular. Este método ayuda a tener una aproximación de las secuencias necesarias para contar con una solución codificada en el lenguaje de programación.

Importante: los diagramas ofrecen un acercamiento simple y visual del flujo de datos de un proceso, mediante un sistema de tratamiento de la información.

Gracias a ellos es posible planear y examinar los pasos requeridos para que dicho flujo o proceso funcione de manera apropiada. Igualmente, ayudan a identificar problemáticas en las operaciones y comprender los procedimientos necesarios para cumplir con el flujo. Por otro lado, permiten analizar las etapas de los diferentes procesos con el fin de optimizarlos y mejorarlos.

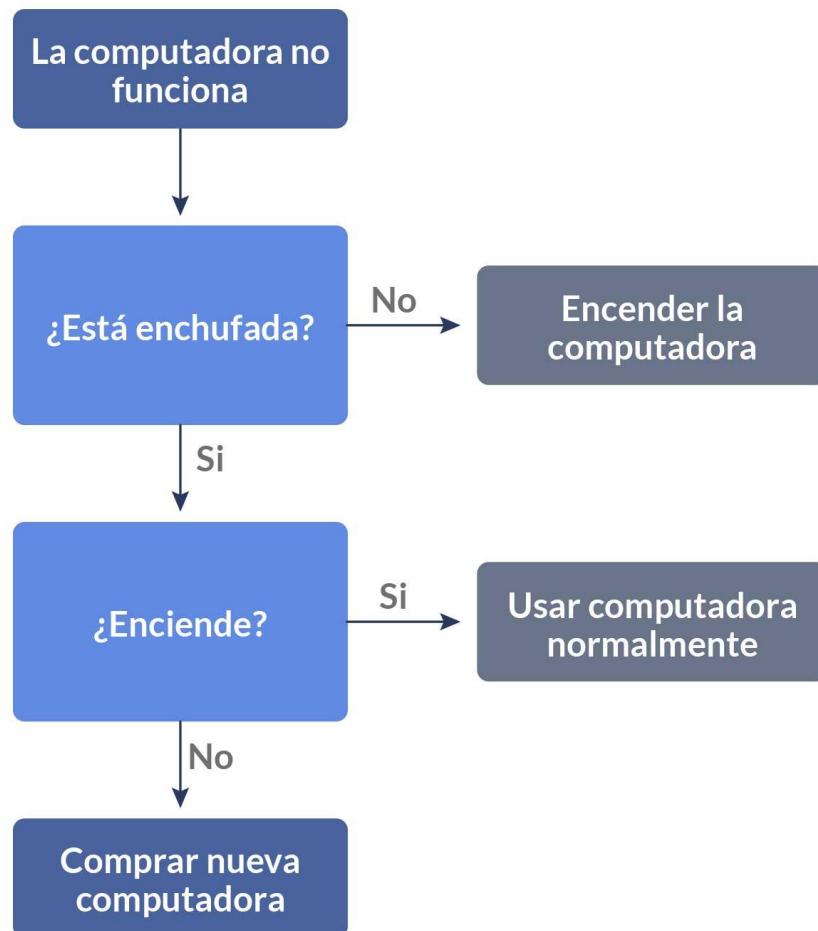
Figura 47. Procesos del diagrama de flujo

A continuación, se definen los símbolos más utilizados para diseñar y construir un diagrama de flujo, utilizados también en la generación de ciclos, bucles y condicionales. Esto te permitirá interpretar y analizar los diagramas de flujo para solucionar eficazmente los problemas algorítmicos que encontrarás más adelante.

Figura 48. Símbolos usados en un diagrama de flujo

Símbolo terminal	Este símbolo indica el inicio o el final de un programa o subprocesso. Generalmente, contiene la palabra inicio o fin.
Símbolo línea de flujo	Muestra el orden de operación de los procesos. Se representa con una línea saliendo de un símbolo y apuntando a otro.
Símbolo de proceso	Este símbolo representa un conjunto de operaciones que realiza el cambio de un valor o la ubicación de un dato.
Símbolo de decisión	Este símbolo es una operación condicional que plantea dos caminos a tomar. La operación utiliza por lo regular una pregunta sí/no o falso/verdadero.
Símbolo de entrada	Este símbolo representa la entrada de datos a un proceso.
Símbolo de salida	Este símbolo representa la salida de datos de un proceso y muestra el resultado del mismo.
Símbolo de conector de página	Este símbolo es usado para conectar diagramas en la misma página mediante el uso de etiquetas y eliminar así líneas extensas que puedan volver el proceso difuso o complejo.
Símbolo de conector fuera de página	Este símbolo es usado para conectar diagramas en páginas diferentes mediante el uso de etiquetas, con el fin de eliminar líneas extensas que puedan volver el proceso difuso o complejo.

Figura 49. Ejemplo de diagrama de flujo



3.3. Análisis de algoritmos

Este término se refiere al conjunto de procesos necesarios para definir y encontrar el nivel de complejidad de un algoritmo que resuelva un problema dado, con el fin de generar estimaciones teóricas de los recursos necesarios para solucionar dicho problema. Generalmente, los conceptos utilizados para definir la complejidad son: tiempo (complejidad temporal) y almacenamiento (complejidad espacial).

La complejidad temporal propone generar una función que determine, con base al flujo de datos y a la cantidad de procesos que tiene el algoritmo, cuánto tiempo se demorará en ejecutarse. Por su parte, la complejidad espacial define la cantidad de espacio o memoria necesaria para almacenar toda la información del proceso.

Dado que este tema es algo amplio y merece un poco más de atención, te invitamos a profundizar sobre la complejidad algorítmica observando el siguiente video:

Video



- **Autor:** Manuel González
- **Título:** Python - Nivel 23 - Reto 1 - Complejidad de un algoritmo
- **URL:** <https://www.youtube.com/watch?v=GD254Gotp-4>

El video anterior destaca la relevancia de la complejidad algorítmica en la actualidad para realizar codificaciones elementales o complejas, dependiendo de la problemática a solucionar, profundizando en las bases de la algoritmia y acerca de cómo usarlas de manera sencilla.

3.4. Herramientas para la elaboración de algoritmos

Para el propósito de este curso, usaremos PSeInt como herramienta para la elaboración de algoritmos, ya que está pensada principalmente para estudiantes, generando pseudocódigo de manera intuitiva y asistiendo al usuario con comandos que facilitarán su trabajo.

Te invitamos a observar los siguientes videos para familiarizarte con PSeInt y conocer más sobre su manejo.

Video



Autor: Danisable

- **Título:** ¡Aprende a programar desde cero con PseInt! | ¿Qué es PseInt?
- **URL:** <https://www.youtube.com/watch?v=FvibfpSVFBw&list=PLAzISdU-KYwXlIXcUCW-ByIQZemcDV798>

Video



- **Autor:** THE SEGA RED
- **Título:** Curso de Programación en PSEINT [#1] - Introducción al curso
- **URL:**
https://www.youtube.com/watch?v=nMoFtd4ln28&list=PL4Wdg_psRY0ZEzlAagli6Lm-q_vXi_arU

En los recursos anteriores observaste la importancia que tienen los software para realizar pseudocódigo, diagramas de flujo y depuración del mismo, dependiendo de la problemática que deseas solucionar, te lleva a desarrollar pseudocódigo óptimo de una manera simple y permite aprender sobre los comandos esenciales para utilizar dicha herramienta (ITSON ,2016).

3.5. Ejercicios prácticos

Antes de finalizar, exploremos algunos ejercicios prácticos sobre la generación de un algoritmo, desde la descripción de la problemática hasta la implementación del pseudocódigo, incluyendo también la elaboración del diagrama de flujo correspondiente.

Presta atención a estos ejercicios, pues te serán de ayuda para desarrollar la evidencia de aprendizaje.

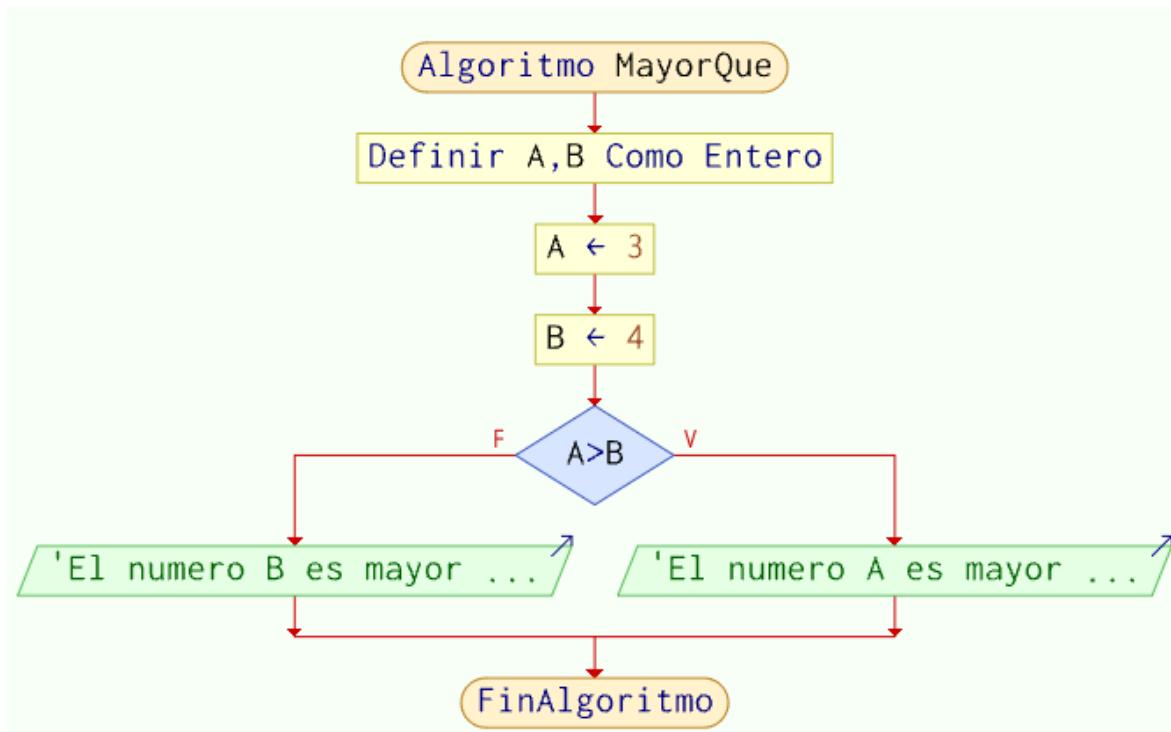
Ejercicio 1

Diseña un algoritmo que permita leer 2 valores distintos y determinar cuál de los dos números es mayor.

Figura 50a. Ejercicio 1 – Algoritmo

```
1  Algoritmo MayorQue
2      Definir A,B Como Entero;
3      A<-3;
4      B<-4;
5      Si A>B Entonces
6          Escribir "El numero A es mayor que el B" ;
7      SiNo
8          Escribir "El numero B es mayor que el A";
9      Fin Si
10     FinAlgoritmo
```

Figura 50b. Ejercicio 1 - Diagrama de flujo

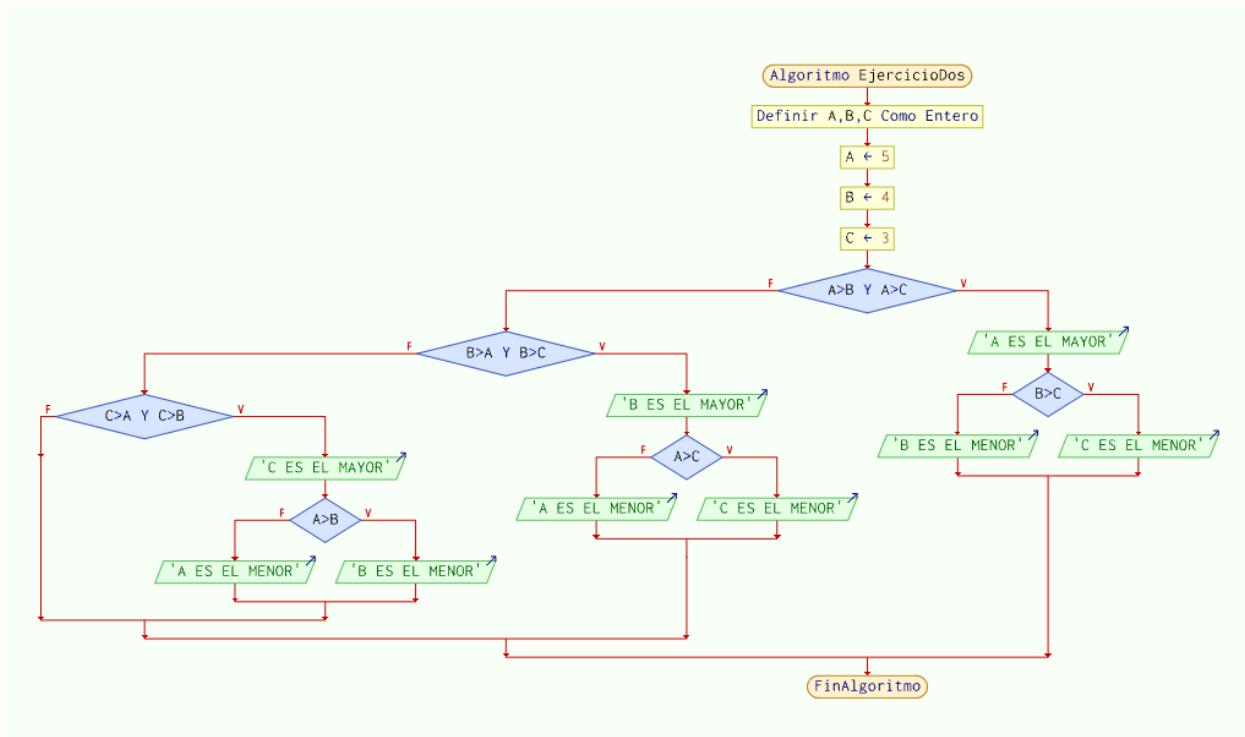


Ejercicio 2

Elabora un algoritmo que permita leer 3 valores y almacenarlos en tres variables: A, B y C. El algoritmo debe determinar cuál de ellos es el valor mayor y cuál es el menor.

Figura 51a. Ejercicio 2 – Pseudocódigo

```
1  Algoritmo EjercicioDos
2      Definir A,B,C Como Entero;
3      A<-5;
4      B<-4;
5      C<=3;
6      Si A>B Y A>C Entonces
7          Imprimir "A ES EL MAYOR"
8          Si B>C Entonces
9              Imprimir "C ES EL MENOR"
10         SiNo
11             Imprimir "B ES EL MENOR"
12         Fin Si
13     SiNo
14         Si B>A Y B>C Entonces
15             Imprimir "B ES EL MAYOR"
16             Si A>C Entonces
17                 IMPRIMIR "C ES EL MENOR"
18             SiNo
19                 IMPRIMIR "A ES EL MENOR"
20             Fin Si
21         SiNo
22             Si C>A Y C>B Entonces
23                 Imprimir "C ES EL MAYOR"
24                 Si A>B Entonces
25                     Imprimir "B ES EL MENOR"
26                 SiNo
27                     IMPRIMIR "A ES EL MENOR"
28                 Fin Si
29             Fin Si
30         Fin Si
31     Fin Si
32 FinAlgoritmo
~~
```

Figura 51b. Ejercicio 2 - Diagramas de flujo

Cierre

Al finalizar el estudio de esta primera unidad, el estudiante estará en capacidad de analizar procesos y generar el respectivo diagrama de flujo y/o pseudocódigo de este, podrá analizar los diferentes flujos y las estructuras de datos que se generan en un algoritmo, estos conocimientos adquiridos le permitirán generar códigos de procesos más complejos aplicarlos en sus actividades del día a día tales como las acciones que realizas para encender y apagar un computador, para mejorar y optimizar procesos, y tener la posibilidad de detectar procesos críticos y determinar tiempos de ejecución de los mismos.

En la siguiente unidad podrás aplicar este conocimiento para mejorar y optimizar las actividades que realizas a diario y adquirir un mayor dominio de las estructuras, estilos y formas de codificaciones.

Referencias de imágenes

- Pixabay. Imagen de Clker-Free-Vector-Images en Pixabay. Recuperado de: <https://pixabay.com/es/vectors/puesto-de-trabajo-computadora-303940/>



Lecturas y Material Complementario

- **Título:** Conozca cuáles son los beneficios de aprender a programar
Autor: Tecnósfera El Tiempo
URL: <https://www.eltiempo.com/tecnosfera/novedades-tecnologia/beneficios-de-aprender-a-programar-40497>
- **Título:** Tutorial de Python
Autor: Python
URL: <https://docs.python.org/es/3/tutorial/>

Recurso interactivo

- **Título:** Learn Python
Autor: Codecademy
URL: <https://www.codecademy.com/learn/learn-python/modules/learn-python-python-syntax-u-6>

Videos

- **Título:** Curso Python para principiantes
Autor: Fazt
URL: <https://www.youtube.com/watch?v=chPhlsHoEPo>
- **Título:** Python - Nivel 23 - Reto 1 - Complejidad de un algoritmo
Autor: Manuel González
URL: <https://www.youtube.com/watch?v=GD254Gotp-4>
- **Autor:** Danisable
Título: ¡Aprende a programar desde cero con PseInt! | ¿Qué es PseInt?
URL: <https://www.youtube.com/watch?v=FvibfpSVFBw&list=PLAzISdU-KYwXlIXcUCW-ByIQZemcDV798>
- **Autor:** THE SEGA RED
Título: Curso de Programación en PSEINT [#1] - Introducción al Curso
URL: https://www.youtube.com/watch?v=nMoFtd4ln28&list=PL4Wdg-psRY0ZEzlAaglj6Lm-q_vXi_arU



Bibliografía

- Baños, Y., & Hernández, A. (2012). *Algoritmos* [Diapositivas]. Universidad Autónoma del Estado de Hidalgo.
https://www.uaeh.edu.mx/docencia/P_Presentaciones/prepa1/algoritmos.pdf
- Cadavid, S., Osorio, A., Chiquito, J., Valencia, L., Marín, J., Arboleda, W., Galeano, J., & García, E. (2021). *Los lenguajes de programación*. Colombia Aprende.
https://aprende.colombiaaprende.edu.co/sites/default/files/naspublic/curriculos_ex/n1g10_fproy/nivel1/programacion/unidad1/leccion1.html
- González, X. (2019). *Diagrama de flujo y su relación con la vida cotidiana* [Tesis de grado - Universidad técnica de Machala]. Repositorio Utmachala.
http://repositorio.utmachala.edu.ec/bitstream/48000/14847/1/E-4389_GONZALEZ%20ESPINOSA%20JENNIFER%20Xiomara.pdf
- ITSON. (2016). *Pseudocódigo y PSEINT*.
https://www.itson.mx/oferta/isw/Documents/guia_pseint_2016.pdf
- Valdés, L. (2018). Simbología. En *Manual para la diagramación de procesos*.
http://docencia.fca.unam.mx/~lvaldes/cal_pdf/cal18.pdf

Esta licencia permite a otros distribuir, remezclar, retocar, y crear a partir de esta obra de manera no comercial y, a pesar que sus nuevas obras deben siempre mencionar a la IU Digital y mantenerse sin fines comerciales, no están obligados a licenciar obras derivadas bajo las mismas condiciones.



IUDigital
de Antioquia
INSTITUCIÓN UNIVERSITARIA
DIGITAL DE ANTIOQUIA