Davante

1

PROGRAMACIÓN

# Introducción a la programación

# ÍNDICE

# ÍNDICE

# OBJETIVOS

*Diferenciar entre datos e información.*

*Conocer la estructura de un programa.*

*Diferenciar los distintos tipos de lenguajes de programación.*

*Comprender el concepto de algoritmo.*

*Realizar el diseño y la representación de algoritmos.*

*Conocer e identificar las diferentes fases del ciclo de desarrollo de software.*

*Conocer los distintos entornos de desarrollo software.*

*Realizar la instalación de un entorno de desarrollo integrado.*

## / 1. Introducción y contextualización práctica

Gracias a la evolución del computador, es posible realizar tareas complejas más rápidamente. Actualmente, vivimos un proceso de digitalización y automatización que influye en el modelo productivo de las empresas y los países.

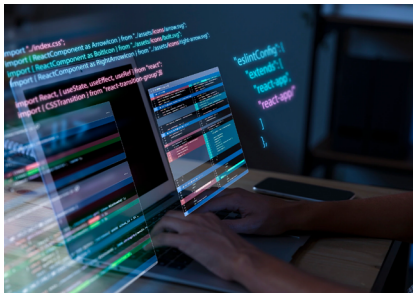

*Fig.1. Código fuente.*

El desarrollo software requiere de una serie de fases que hay que tener en cuenta para garantizar la obtención de los resultados deseados.

Durante este tema, se define el concepto de algoritmo para dar solución a diferentes tipos de problemas.

No existe una regla general para poder crear algoritmos de una manera sencilla y esquematizada, pues para ello la única forma es practicar creándolos. A medida que vayamos elaborando diferentes tipos de algoritmos, veremos como la dificultad a la que nos enfrentamos disminuye.

Sin embargo, existen circunstancias en las que no es suficiente con la práctica y tenemos que atender a técnicas más especializadas para la creación de algoritmos.

A continuación, se presenta un caso práctico a través del cual podremos aproximarnos de forma aplicada a la teoría de este tema.

*Audio intro. "Validación de algoritmos"*

*https://on.soundcloud.com/LXX5rnX16aT2Jxw17*

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema. Encontrarás su resolución en el apartado 'Resumen y resolución del caso práctico'.

# / 2. ¿Qué es la informática?

Podemos definir la informática como el conjunto de conocimientos, tanto científicos como técnicos, que permiten el procesamiento auto**mático** de la **infor**mación mediante la utilización de ordenadores. De ahí el término '**informática**'.

En esta definición han aparecido dos términos claves que vamos a desarrollar:

- Información.
- Ordenador.

## 2.1. ¿Qué es la información?

A la hora de definir el concepto de información es necesario distinguir entre datos e información:

**Datos**: Pueden ser cualquier tipo de información, como números, palabras, imágenes, sonidos, entre otros, que, de manera individual, tiene un significado puntual y se emplea para la representación de algún hecho, concepto o entidad real.

**Información**: Mientras que los datos tienen un significado puntual, el resultado del procesamiento, la interpretación y la interrelación de los mismos da lugar a la información. Dependiendo de la interrelación, se puede conseguir una información u otra. Por ejemplo, imaginemos los datos de los trabajadores de una empresa:

- Nombre.
- Edad.
- Estudios.
- Salario.

De manera individual, representan valores, sin embargo, una vez organizados por edad y salario, se obtiene un informe sobre la distribución del sueldo en función de la edad. Esta organización puede generalizarse a otros campos, como pueden ser los estudios y el salario, dando lugar a una nueva información que nos indica la distribución del salario en función de la formación de los empleados.

**Como no vamos a tratar específicamente sistemas de gestión de información, consideraremos datos e información**

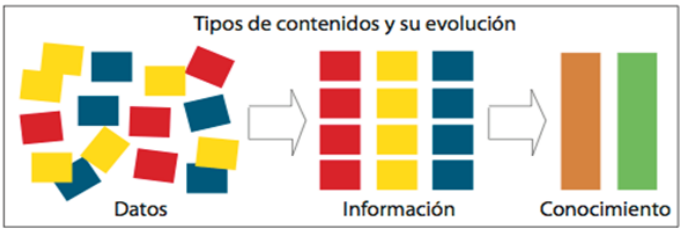


*Fig.2. Datos, información y conocimiento.*

**como sinónimos**.

Así como la información deriva de los datos, de la información deriva el conocimiento. Sin embargo, en esta fase de construcción del conocimiento son las personas las encargadas de realizar el trabajo.

## 2.2. Procesamiento de la información

Se puede definir un ordenador como una máquina que es capaz de procesar la información realizando algunas transformaciones solicitadas por el usuario. En este contexto, es importante destacar que el usuario puede ser una persona o bien otro programa informático.

Al hablar de **máquina de proceso**, tenemos que distinguir diferentes elementos:

- **Entrada**: Hace referencia a los elementos que se introducen en la máquina y de los que depende el resultado.
- **Unidad de proceso**: Es el elemento en el que se realiza el procesamiento de los datos de entrada.
- **Salida**: Es el resultado de procesar los datos de entrada en la unidad de procesamiento.

> ### ¡! Sabías que...
>
> La máquina de Turing es un modelo teórico de computación que formaliza el concepto de algoritmo y cálculo. El impacto de esta máquina se extiende a varias áreas de la informática y la matemática.

Durante el procesamiento de los datos de entrada, se realizan una serie de transformaciones en la unidad de proceso que dependen de lo implementado por el programador. De esta forma, ante los mismos datos de entrada, si se varían las transformaciones realizadas, el resultado es diferente. Una peculiaridad de la informática es que **los resultados siempre son deterministas**, pues ante la misma entrada y aplicando las mismas transformaciones, el resultado debe ser el mismo.

De esta forma, hablamos de **proceso** cuando nos referimos a un conjunto de acciones que tienen que llevarse a cabo en tiempo de ejecución. Un proceso es ejecutado dentro del procesador, siendo este último parte de la unidad de procesamiento de un computador. Normalmente, decimos que un proceso está compuesto de instrucciones que tienen que ser ejecutadas en un orden determinado para obtener el resultado deseado.

## 2.3. Hardware vs. software

Por tanto, se puede decir que, en toda solución informática, tenemos que distinguir dos grandes grupos de elementos:

- **Hardware**: Hace referencia a todos los elementos físicos y **tangibles** que conforman un computador.

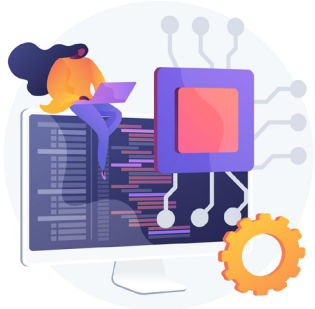

*Fig.3. Hardware vs. software.*

- **Software**: Normalmente, hace referencia a todo aquello **intangible** que forma parte de una solución informática.

Tradicionalmente, se ha dado más importancia al hardware que al software. Sin embargo, en los últimos años, el software adquiere un papel fundamental, siendo incluso más relevante que el hardware utilizado para ejecutarlo.

# / 3. Caso práctico 1: "Datos vs. información"

**Planteamiento**: La diferencia fundamental entre datos e información radica en su significado y contexto. Mientras

| Producto | Producto 1 | Producto 2 |
|---|---|---|
| Nombre | *Smartphone* | *Laptop* |
| Modelo | Galaxy S23 | Mackbook Pro |
| Precio | 999 € | 1.680 € |
| Ventas | 20 | 10 |
| Cantidad en *stock* | 30 | 16 |

*Tabla 1. Inventario.*

los datos son hechos objetivos, observaciones o medidas sobre el mundo que nos rodea y se presentan en forma cruda y sin procesar, la información se crea al organizar, analizar o interpretar los datos para obtener significado, así que es el resultado de procesar estos datos de manera que sean útiles para la toma de decisiones o acciones específicas.

**Nudo**: Imagina que estás administrando el inventario de una tienda de electrónica cuya base de datos almacena información sobre los productos en existencia. ¿Qué información se puede sacar de los siguientes datos?

**Desenlace**: La información es el resultado de un análisis sobre el estado del inventario:

El producto con más ventas en el inventario es el *smartphone* Galaxy S23, ya que se han vendido 20 unidades y es el producto con mayor cantidad en *stock* disponible.

Esta información se ha generado al analizar los datos disponibles: nombres de productos, modelos, precios, ventas y cantidades en *stock*.

# / 4. Programa informático

Hablamos de programa informático cuando nos referimos tanto a los **datos** como a la **secuencia de acciones** que hay que llevar a cabo para realizar el procesamiento deseado.

De esta definición podemos observar que un **programa informático** es la unidad en reposo de las instrucciones del algoritmo implementado. Sin embargo, un **proceso** es un programa informático que pasa a la unidad de

procesamiento para ser ejecutado.

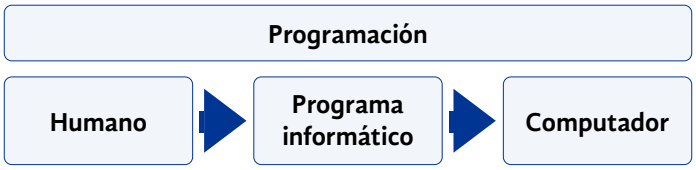


*Fig.4. Comunicación entre humano y computador.*

Por tanto, se puede definir un programa como una serie de instrucciones que indican, de forma precisa y exacta, al

```
class HolaMundo {
        public static void main(String[] args{)
                System.out.println("Hola mundo");
        } // Fin método main
} // Fin class HolaMundo
```

computador, qué tiene que hacer.

En esta relación, el programa es el nexo de unión entre la máquina encargada de emplear sus capacidades para resolver un problema y los humanos que requieren que la máquina solvente dicho problema.

La ciencia y el arte de la programación consisten en saber construir **un modelo de solución** para un problema dado y en indicar una serie de **instrucciones** que permitan describir dicho modelo al computador.

Como ejemplo de programa, veamos un caso sencillo en Java

Cuando el programa se ejecute, imprimirá la siguiente línea de texto: Hola mundo.

# / 5. Lenguajes de programación

Es posible utilizar diferentes lenguajes de programación **para establecer una comunicación con una computadora y poder crear programas que puedan ejecutarse en el procesador**. Como norma general, es posible diferenciar tres grandes grupos de lenguajes de programación: lenguaje máquina, lenguaje ensamblador y lenguajes de alto nivel.

## 5.1. Clasificación

Entre los lenguajes de programación, encontramos:

- **Lenguaje máquina**: Se trata de un lenguaje directamente entendible por el procesador, pues está compuesto por las instrucciones que este es capaz de entender. Se trata de instrucciones basadas en 0s y 1s. El lenguaje máquina puede variar dependiendo del procesador en cuestión, por lo tanto, **es diferente de una computadora a otra**. Por ejemplo, uno de los elementos que pueden hacer cambiar el lenguaje máquina es la **memoria principal**. En este caso, el tamaño de la memoria puede hacer que se haga referencia a posiciones que no existan en alguna computadora. De esta forma, se puede observar que el lenguaje máquina es muy dependiente de la

plataforma que se está utilizando. Además, para cualquier programador, resulta extremadamente complicado programar con esta codificación.

- **Lenguaje ensamblador**: Surge para intentar **simplificar la complejidad del lenguaje máquina**. Aun así, se considera un lenguaje de bajo nivel, en ocasiones confundido con el lenguaje máquina. En el lenguaje ensamblador se utilizan **comandos**, normalmente de longitud tres para realizar las diferentes operaciones.

> **Recuerda...**
>
> No existe un lenguaje de programación válido para todos los contextos. Es necesario realizar una correcta elección del lenguaje de programación y adaptarlo al problema y contexto en el que vamos a trabajar.

*Fig.5. Código máquina.*

Por ejemplo, se puede emplear el comando SUB para hacer restas. Estos comandos esperan tres o cuatro argumentos. En el ejemplo anterior, en el caso de la resta, se esperan tres argumentos además del comando de la operación.

- **Lenguaje de alto nivel**: Dada la complejidad que tienen tanto el lenguaje máquina como el ensamblador, surge la necesidad de **facilitar el desarrollo de programas**. Con este fin, se crean los lenguajes de programación de alto nivel. Una de las ventajas de los lenguajes de programación de este tipo es su **similitud con el lenguaje natural humano**, normalmente con el inglés. Sin embargo, estos lenguajes no pueden ser ejecutados directamente por el procesador, pues este solo entiende lenguaje máquina. Es en este punto donde se introduce el concepto de **compilación/interpretación**, es decir, una traducción a lenguaje máquina.

## 5.2. Estilo de programación

Según el estilo de programación, diferenciamos:

- **Lenguajes imperativos o procedimentales**: Se centran en **cómo se debe realizar una tarea**, detallando los pasos exactos que el programa debe seguir para alcanzar un resultado y el uso de variables para almacenar valores con los que operará.

- **Lenguajes declarativos**: En ellos **se describe el resultado deseado sin especificar los pasos exactos** para llegar a él. Hay dos tipos de lenguajes declarativos:

  - **Lenguajes funcionales**.

  - **Lenguajes de programación lógica**.

## 5.3. Traducción de lenguajes

Es el proceso de convertir un programa escrito en un lenguaje de programación particular en otro lenguaje de programación, manteniendo su funcionalidad y semántica. Esto puede involucrar diferentes niveles de abstracción y complejidad, dependiendo de los lenguajes de origen y destino. Según el método que se use para llevar a cabo

la **traducción**, se habla de lenguajes compilados o lenguajes interpretados. Hay dos tipos básicos de traductores,

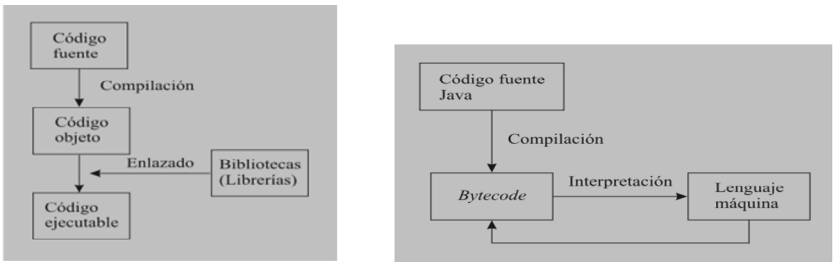


*Fig.6. Proceso de compilación vs. interpretación.*

dependiendo de cómo se hace la traducción:

- **Compiladores**: Son programas que traducen los programas fuente escritos en lenguajes de alto nivel (Pascal, Cobol, C) a lenguaje máquina. El proceso es el siguiente: traducen el programa fuente completamente a programa objeto, y cuando han terminado hacen el 'linkado' y ejecutan el programa.

- **Intérpretes**: Ejecutan cada instrucción según las va traduciendo (con 'linkado' intermedio si fuera necesario). Si hay un error, no dejan llegar al final, paran en ese error. Son mucho más lentos, pero tienen la ventaja de ser más didácticos.

En la siguiente imagen podrás ver el proceso de complicación e interpretación.

### 5.4. Sintaxis y semántica

Se trata de dos conceptos fundamentales en todo lenguaje de programación. Se corresponden con el **qué puede hacer el lenguaje de programación y** el **cómo indicar que lo haga**.

- **Semántica**: Es el estudio del significado de las unidades lingüísticas.

- **Sintaxis**: Es la manera de enlazarse y ordenarse las palabras en la oración o las oraciones en el periodo.

La sintaxis se refiere a las reglas y estructuras gramaticales que definen cómo se deben escribir las instrucciones en

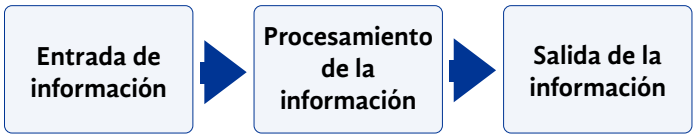


*Fig.7. Estructura de un programa.*

un lenguaje de programación, mientras la semántica alude al significado o interpretación del código escrito en un lenguaje de programación.

## / 6. Estructura de un programa

Los programas se encargan de realizar varias tareas para alcanzar sus objetivos, y para llevarlas a cabo se basan en el uso de **algoritmos**.

Todos ellos siguen típicamente un modelo básico de entrada, procesamiento y salida de información, como muestra la Figura 7.

Todo programa acepta información de entrada (los datos) y la procesa por medio de uno o varios algoritmos. El

resultado constituye la información de salida que es la que vamos buscando.

Respecto al procesamiento, este se mide en términos de **eficacia** y **eficiencia**. Se trata de dos consideraciones clave en el diseño y la implementación de programas. Estos dos conceptos son importantes y deben ser claramente distinguidos por el programador.

- **Eficacia**: Implica que el programa debe cumplir con los requisitos funcionales especificados y satisfacer las necesidades del usuario de manera precisa, es decir, **que el programa funcione**.

- **Eficiencia**: Se refiere a la capacidad de un programa para realizar su trabajo de manera rápida y utilizando la menor cantidad posible de recursos (tiempo, memoria central o de disco), es decir, **que el programa funcione bien**.

Tanto la eficacia como la eficiencia son aspectos importantes a considerar al desarrollar programas. Un programa debe ser efectivo para cumplir con los requisitos del usuario y eficiente para llevar a cabo su trabajo de manera rápida y con recursos mínimos. Un equilibrio adecuado entre estos dos aspectos es esencial para crear programas de alta calidad y rendimiento.

Por lo que respecta a un programa cualquiera, con sus componentes de entrada-procesamiento-salida, se puede organizar como un solo bloque monolítico de código o modularizarlo (subdividirlo) en varios bloques más pequeños.

En un **programa monolítico** solo existe un bloque, un solo programa principal (denominado, en inglés, *main*).

Un **diseño modular**, descendente (*top-down*), se basa en la realización de una serie de descomposiciones sucesivas del algoritmo inicial, que describen el repertorio de instrucciones que van a constituir el programa. Un programa quedará formado por una serie de módulos, cada uno de los cuales realiza una parte concreta de la tarea total.

# / 7. Algoritmos

El concepto de algoritmo proviene de la matemática y se refiere a una descripción precisa de cómo realizar una determinada tarea utilizando un lenguaje muy preciso y descriptivo sin ambigüedades.

La definición de un algoritmo debe describir tres partes: entrada, proceso y salida. Esta definición da lugar a confusiones entre un algoritmo y un programa informático, sin embargo, existen muchas diferencias entre ellos:

Un **algoritmo** se define como un conjunto **finito**, ordenado, de reglas o instrucciones bien definidas, de modo que, siguiéndolas paso a paso, se obtiene la solución a un **problema dado**, mientras que un **programa** soluciona un **problema general** (**determinista**).

De esta forma, un programa informático podría estar compuesto por múltiples algoritmos que permitan la resolución de problemas muy concretos. Uniendo múltiples algoritmos podemos dar solución a un problema de mayor envergadura.

## 7.1. Complejidad de los algoritmos

Para resolver un problema, se pueden utilizar muchos algoritmos. Lógicamente, siempre se debe intentar seleccionar

el mejor. La pregunta es clara: ¿cómo sabemos cuál es el mejor algoritmo? El problema se soluciona a partir de los conceptos de **eficacia y eficiencia**.

| Algoritmo | Complejidad | 1 segundo |
|---|---|---|
| 1 | n | 1.000 |
| 2 | n logn | 140 |
| 3 | $n^2$ | 32 |
| 4 | $n^3$ | 10 |
| 5 | $2^n$ | 10 |

*Tabla 2. Complejidad de un algoritmo.*

Todo algoritmo debe ser eficaz (debe resolver el problema), pero no todos son igualmente eficientes (no usan los mismos recursos para resolver el problema).

¿Cómo se determina la eficiencia de un algoritmo? Este es el verdadero problema, ya que se puede medir de diferentes maneras, dependiendo de qué aspecto de la complejidad estemos considerando: tiempo, espacio o estructura, siendo los más comunes la complejidad temporal y espacial.

La eficiencia se representa como una función que depende de un parámetro llamado tamaño 'n'. La complejidad de un algoritmo se llama orden O(n).

A continuación, se muestra una tabla con cinco algoritmos que tienen distinta complejidad y, por tanto, distinta capacidad de procesamiento (los elementos se pueden procesar en un segundo):

## 7.2. Características de los algoritmos

Un algoritmo reúne ciertas características que tenemos que considerar para que se trate de un algoritmo válido desde el punto de vista informático:

- Debe ser **preciso** e indicar el orden de realización de cada paso.
- Ha de estar **definido**.
- Siempre produce los mismos resultados para los mismos datos de entrada, es decir, si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez (**determinismo**).
- Debe ser **finito**, es decir, debe tener un número finito de pasos.
- Es **independiente** del lenguaje de programación.

De entre todas las características, es necesario destacar las dos fundamentales: finitud y determinismo.

Para poder definir un algoritmo que solucione un problema correctamente, es necesario tener claro lo que queremos solucionar. Aunque esto parece algo obvio, en la realidad, cuando nos enfrentamos a la resolución de un problema en un proyecto, puede no serlo tanto. Normalmente, esto se debe a fallos en fases previas como las de análisis y diseño.

En este punto, podemos introducir el concepto de **semialgoritmo**. Los semialgoritmos son aquellas soluciones

*Audio 1. "Tipos de errores"*
*https://on.soundcloud.com/ZebXuodVAeYYTp5QA*

parciales que pueden no satisfacer las dos características anteriores de finitud y determinismo. De esta forma, podemos decir que todo programa informático debe ser al menos un semialgoritmo, ya que, como mínimo, siempre va a terminar su ejecución, aunque no produzca un resultado.

Existen ocasiones en las que un programa informático puede no generar ningún resultado y además no finalizar su ejecución. En este contexto, es cuando podemos hablar de un error de codificación o ***bug***. Cuando se produce un error de este tipo, es necesario generar una nueva versión del software que arregle el problema.

Normalmente, esto se realiza mediante parches de actualización que pueden ser autoinstalados en el sistema.

| Símbolo de diagrama de flujo | Nombre | Descripción |
|---|---|---|
|  | Símbolo de proceso | Representa un proceso, una acción o una función. |
|  | Símbolo de inicio y fin | Representa el punto de inicio, el punto de fin y los posibles resultados de un camino. |
|  | Símbolo de decisión | Indica una pregunta que debe responderse, por lo general sí/no o verdadero/falso. |
|  | Línea de flujo | Indica el orden de la ejecución de las operaciones. |
|  | Símbolo de entrada y salida | Representa la entrada/salida de datos. |

*Tabla 3. Símbolos de un diagrama de flujo.*

Pueden existir situaciones de mayor vulnerabilidad por este tipo de errores, por ejemplo, fallos de seguridad que pueden hacer que nuestros datos queden expuestos a otros usuarios con malas intenciones. Por este motivo, **es importante asegurarnos de que el software realmente hace lo que debe hacer y de una forma determinista**.

En el siguiente audio podrás aprender más sobre el apartado anterior.

## 7.3. Representación de algoritmos

Antes de empezar a programar, es necesario representar el algoritmo que se va a implementar. Esta representación se refiere a la **forma en que se documenta o describe un algoritmo para que sea comprensible** por otros programadores o para su implementación en código.

- **Diagramas de flujo**: Esta técnica usa representaciones gráficas que muestran la secuencia de pasos y relaciones entre ellos empleando símbolos y flechas. Suele utilizarse en las fases de análisis.

- **Pseudocódigo**: Esta técnica se emplea para describir la lógica y los pasos de un algoritmo de manera clara y concisa, sin preocuparse por la sintaxis específica de ningún lenguaje de programación en particular al emplear el lenguaje natural.

Mediante el pseudocódigo se puede escribir la solución de un problema en forma de algoritmo utilizando palabras y frases del lenguaje natural sujetos a unas determinadas reglas. Se considera como un **paso intermedio entre la solución de un problema y su codificación en un lenguaje**.

Las ventajas de utilizar un pseudocódigo frente al diagrama de flujo son las siguientes:

- Ocupa menos espacio en una hoja de papel.

- Permite representar de forma fácil operaciones repetitivas complejas.

- Es muy fácil pasar de un pseudocódigo a un programa en algún lenguaje de programación.

- Si se siguen las reglas, se puede observar claramente los niveles que tiene cada operación.

# / 8. Caso práctico 2: "¿Cine o parque?"

**Planteamiento**: El pseudocódigo es una herramienta importante en el proceso de desarrollo de software por varias

```
Inicio
    // Pedir al usuario que ingrese dos números
    Escribir "Ingrese el primer número:"
    Leer numero1
    Escribir "Ingrese el segundo número:"
    Leer numero2

    // Sumar los dos números
    resultado = numero1 + numero2

    // Mostrar el resultado
    Escribir "La suma de", numero1, "y", numero2, "es:", resultado
Fin
```

*Fig.8. Resolución del algoritmo.*

razones, como son las siguientes:

- Facilita la comprensión.

- Permite planificar y diseñar algoritmos.

- Fomenta la colaboración.

- Es independiente del lenguaje de programación.

- Ayuda en la enseñanza y el aprendizaje de algoritmos.

**Nudo**: Realiza el pseudocódigo de un algoritmo que sume dos números introducidos por teclado.

**Desenlace**: El pseudocódigo de resolución del algoritmo es el siguiente:

# / 9. Ciclo de vida del software

Describe el proceso que se ha de seguir para la creación o desarrollo de un producto software, ya sea una aplicación de escritorio, una página web o una *app* móvil.

El ciclo de vida del software se divide en varias fases. Hay que ir pasándolas para poder validar el buen desarrollo de nuestro producto software. Es importante ir cumpliendo con todas las fases de desarrollo, ya que un error en alguna de ellas seguramente nos hará retroceder a la anterior, y así constantemente.

La gran ventaja de **seguir el ciclo de vida correctamente es que nos va a permitir detectar los errores que haya en nuestro desarrollo mucho más fácilmente**.

Para cualquier herramienta software, las **etapas** de su ciclo de vida son:

- **Análisis**: Responde a la pregunta ¿qué? Se especifica la lista de requisitos y restricciones acerca de qué va a hacer el programa.

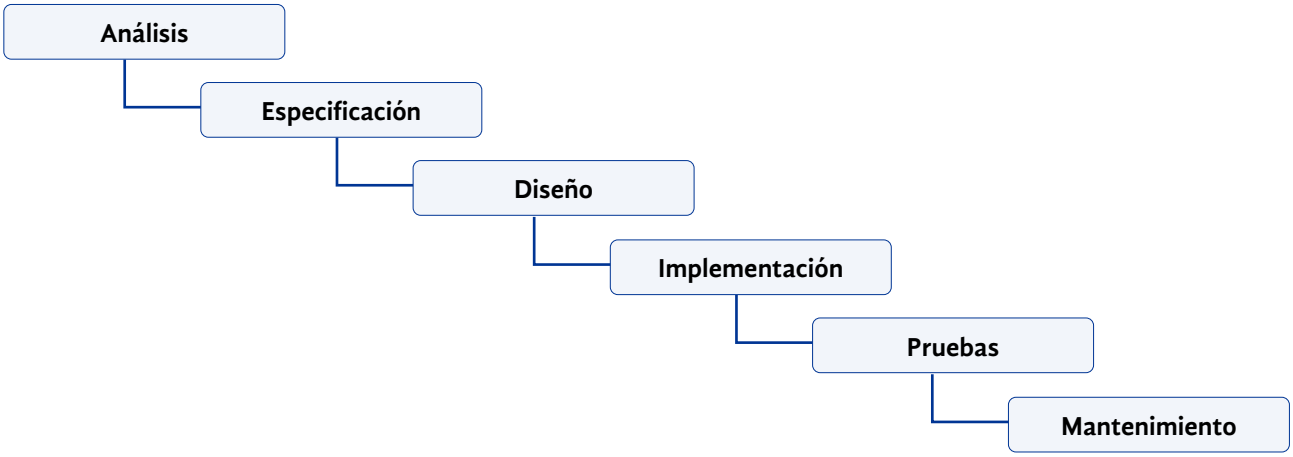


*Fig.9. Ciclo de vida del software.*

### Recuerda...

Un buen análisis y diseño puede suponer la diferencia entre el éxito y el fracaso de un desarrollo software. Muchas veces se menosprecia el diseño, pero puede ser determinante en un proyecto software.

- **Especificación**: Esta fase consiste en la descripción detallada del software haciendo especial hincapié en el comportamiento esperado y su interacción con los usuarios y/o otros sistemas.

- **Diseño**: Responde a la pregunta ¿cómo? En esta fase se van a diseñar los algoritmos necesarios que se representarán mediante el pseudocódigo. Para hacer el diseño se usan los diagramas de clase.

- **Implementación**: Una vez se haya realizado el diseño del programa, es necesario implementarlo mediante la generación de un código fuente que se ajuste a él obteniendo el programa fuente.

- **Pruebas**: A la vez que se implementa, se va probando el software que vamos desarrollando, cuya funcionalidad es encontrar fallos. Suelen hacer pruebas personas ajenas a la implementación. No es probar que funcione, sino buscar dónde puede fallar.

- **Mantenimiento**: Esta etapa corresponde al 80% del ciclo de vida del software y va desde la obtención de una herramienta operativa hasta la retirada del programa, incluyendo la modificación de funcionalidades del software.

# / 10. Herramientas básicas para el desarrollo

Las herramientas necesarias para poder desarrollar software son básicamente tres: **editor de texto plano, compilador y línea de comandos**. El editor de texto nos permitirá escribir el código fuente de nuestro programa de una manera sencilla. Utilizando la línea de comandos podemos llamar al compilador indicando el nombre del archivo que queremos compilar.

A medida que el programa que desarrollamos aumenta de tamaño, se hace necesario dividirlo en diferentes archivos, luego en módulos, componentes, etc. En este contexto, es necesario invocar al enlazador/*linker* para poder obtener el archivo ejecutable. Además, a medida que aumenta el número de líneas de nuestro código, es más factible que podemos introducir algún error en la escritura del programa. Localizar errores de compilación con estas herramientas básicas, si bien es posible, en ocasiones es muy costoso, por lo que cobra sentido utilizar entornos de desarrollo que hayan sido creados para este fin.

## 10.1. Entornos de desarrollo integrados (IDE)

Un entorno de desarrollo integrado se compone de un **conjunto de herramientas software que permiten al desarrollador llevar a cabo diferentes tareas durante la creación del software**. En este tipo de entornos, normalmente,

*Vídeo 1. "Instalación de Netbeans"*
https://bit.ly/3g8SGxv

*Vídeo 2. "Compilación de un proyecto de Netbeans"*
https://bit.ly/2VsiZqK

se incluyen herramientas como el editor de código fuente, el depurador, el compilador, el documentador, etc. La mayoría de los IDE actuales permiten el autocompletado del código fuente, así como el resaltado de posibles errores de compilación. Esto es posible porque realizan una **precompilación** en tiempo real, es decir, mientras se escribe el código fuente.

Con la utilización de IDE **se acelera el desarrollo software**, pues al disponer de todas las herramientas unificadas, se ahorra tiempo al desarrollador, minimizando posibles fuentes de errores. Además, en caso de errores, los IDE suelen ofrecer información adicional para poder solucionarlos fácilmente.

La gran mayoría de IDE incorporan grandes funcionalidades para mejorar la edición del código fuente. Entre estas **características** podemos destacar:

- **Edición de código**: El editor de código de un IDE está diseñado para escribir código fuente. En algunos casos

se incorporan funcionalidades gráficas para facilitar la comprensión del código generado.

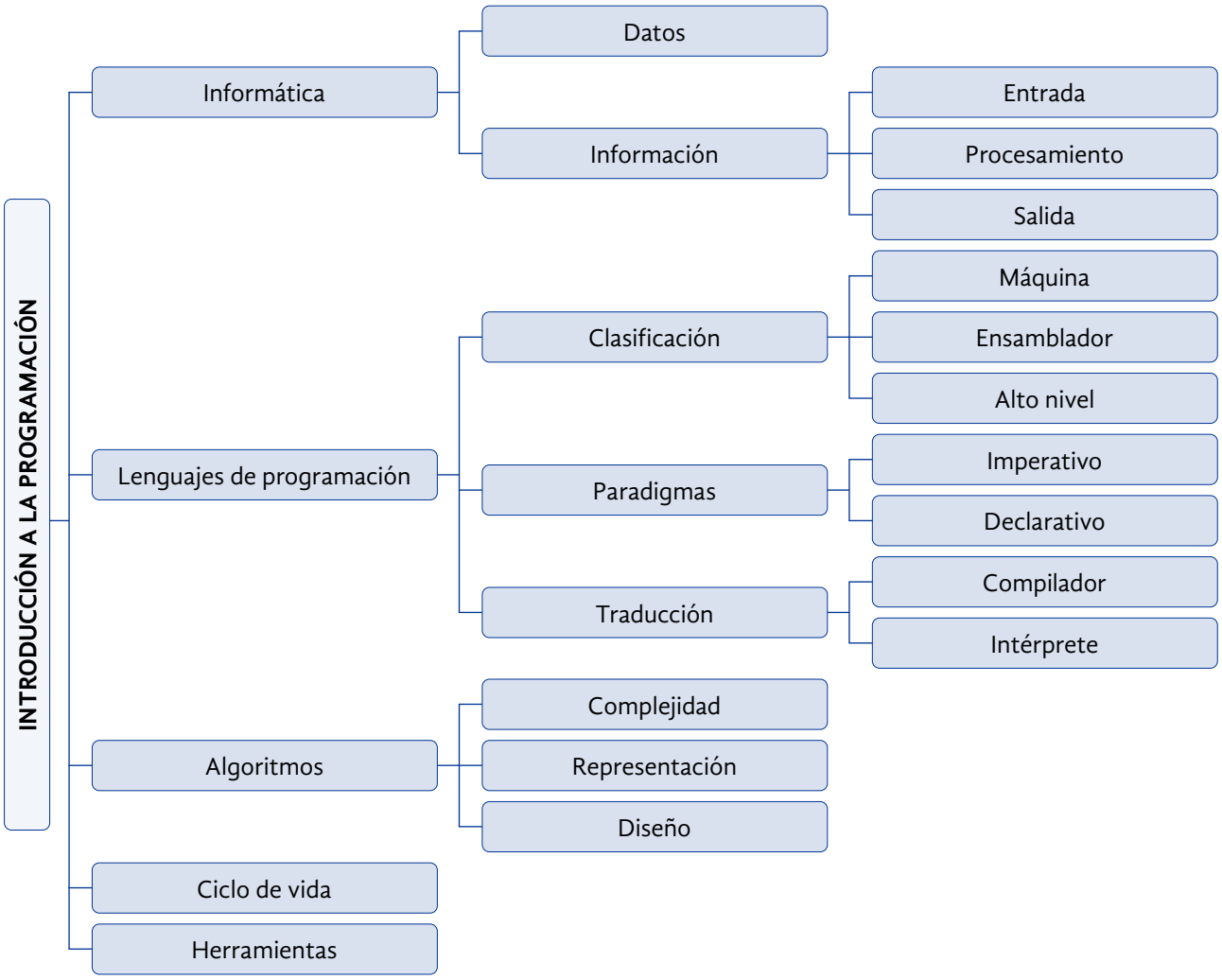


*Fig.10. Esquema resumen del tema.*

- **Resaltado de la sintaxis**: Gracias al resaltado de la sintaxis del código fuente, se pueden identificar fácilmente las diferentes partes del código.

# / 11. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema se ha realizado un primer acercamiento al mundo de la programación. Se ha visto la analogía entre los distintos lenguajes de programación y los lenguajes de comunicación de los humanos, ya que, al fin y al cabo, lo que estamos haciendo es un proceso de comunicación, pero con una máquina en lugar de con otro ser humano, y por ello, nos debemos adaptar a los lenguajes que las máquinas entienden.

Se ha destacado la importancia del campo de la algoritmia y el proceso de ciclo de vida del software para poder realizar una correcta implementación de un proyecto teniendo los menos problemas posibles.

Además, se ha visto que existen muchos entornos de desarrollo integrado en el mercado. Nosotros hemos utilizado

NetBeans por tratarse de un IDE libre no propietario y multiplataforma.

A continuación podrás ver el esquema resumen del tema.

## Resolución del caso práctico de la unidad

Todo sistema informático debe pasar unas pruebas de verificación que permitan asegurar que el software es capaz de realizar aquellas tareas para las que fue diseñado. En este sentido, es necesario garantizar las características básicas de cualquier algoritmo, como finitud y determinismo.

En este caso práctico, podemos ver que los algoritmos implementados carecen tanto de finitud como de determinismo. En lo referente a la finitud, no se cumple, porque, en ocasiones, no se generan resultados, dejando a los aviones a la espera de poder aterrizar sin ninguna pista asignada. Como hemos visto en el tema, es necesario que un algoritmo siempre genere una respuesta. En cuanto al determinismo, vemos que, en este caso, no se cumple, porque, ante las mismas situaciones, los aviones obtienen diferentes pistas para aterrizar. Como hemos visto en el tema, el determinismo nos asegura que, ante las mismas entradas y condiciones, los algoritmos generan el mismo resultado.

# / 12. Bibliografía

ALGORITMIA ALGO+ (s. f.). *Algoritmos y estructuras de datos*. Recuperado de: http://www.algoritmia.net/

Bhargava, A. Y. (2019). *Algoritmos: Una guía ilustrada para programadores y curiosos*. Grupo Anaya.

Caro, C. M., Ramos, A. N. y Barceló, A. V. (2002). *Introducción a la programación con orientación a objetos*. Prentice Hall.

Manuel, P. G. J. (2022). *Entornos de desarrollo*. Ediciones Paraninfo, S. A.

Vorderman, C. (2019). *Introducción a la programación informática: una excelente guía visual que explica, paso a paso, desde el código binario hasta la creación de juegos*. Editorial DK.

Roughgarden, T. (2021). *Algoritmos iluminados (primera parte): Conceptos básicos*. Algoritmos iluminados.

Ojeda, F. C. (2022). *Introducción a la programación*. Anaya Multimedia.