

Programación 1

Año de cursada 1º Año

Clase N.º 1: Introducción a la Programación

Contenido:

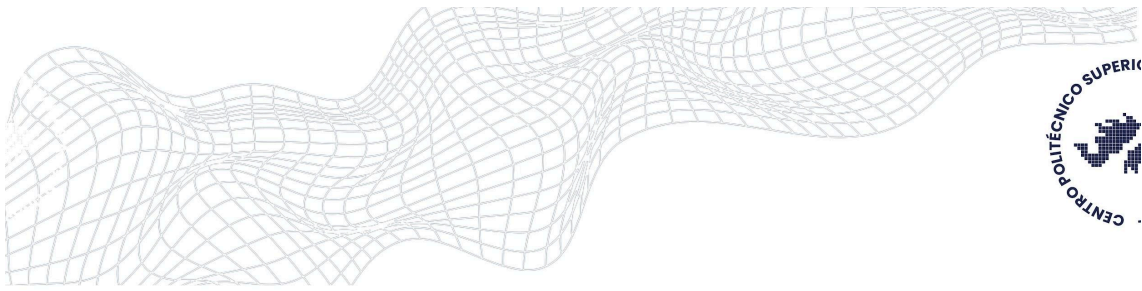
En la clase de hoy trabajaremos los siguientes temas:

- Lenguajes de programación: Historia de los lenguajes de programación, tipos de lenguajes, intérpretes y compiladores.
- Paradigmas de programación: Programación estructurada, modular y programación orientada a objetos.
- Aplicaciones: Programas y aplicaciones, historia del software, crisis del software. El ciclo de vida de una aplicación, Errores.
- Definición y análisis de problemas del campo informático, Definición y análisis de problemas del campo informático.

1. Presentación:

¡Hola, bienvenidos/as a la primera clase de Programación! Esta clase ha sido pensada para los/as recién iniciados/as en el mundo de la programación 1.

En esta clase, exploraremos los fundamentos de la programación. Analizaremos los paradigmas de programación: estructurada,



modular y orientada a objetos, así como los lenguajes de programación y su historia. También abordaremos aplicaciones, la crisis del software, el ciclo de vida de una aplicación, errores y las etapas del desarrollo clásico.



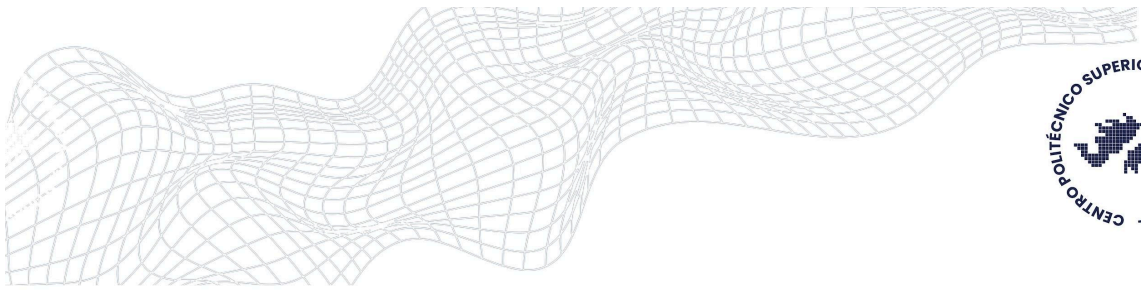
Para comenzar, te invitamos a ver el siguiente video introductorio sobre la evolución de los lenguajes de programación:

https://www.youtube.com/watch?v=MtuqCOL2_S0

Título: Historia de los lenguajes de Programación

Autor: Maestro de la computación





Luego, responde:

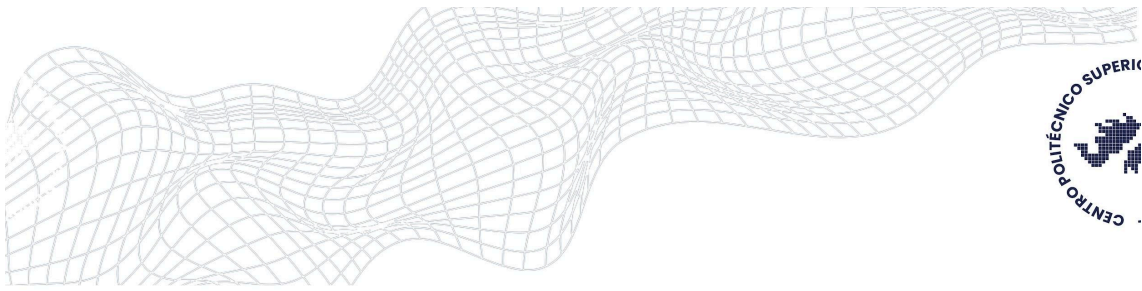
1. ¿Cuál crees que ha sido el avance más importante en los lenguajes de programación?
2. ¿Cómo influyen estos cambios en la forma en que programamos hoy?

2. Desarrollo y Actividades:



Historia de los lenguajes de programación:

Los lenguajes de programación se han desarrollado a lo largo de varias décadas. El primer lenguaje de programación de alto nivel fue FORTRAN, desarrollado en la década de 1950. Desde entonces, se han desarrollado

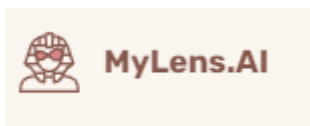


muchos otros lenguajes de programación, incluyendo COBOL, BASIC, C, C++, Java, Python, Ruby, entre otros.



Desafío:

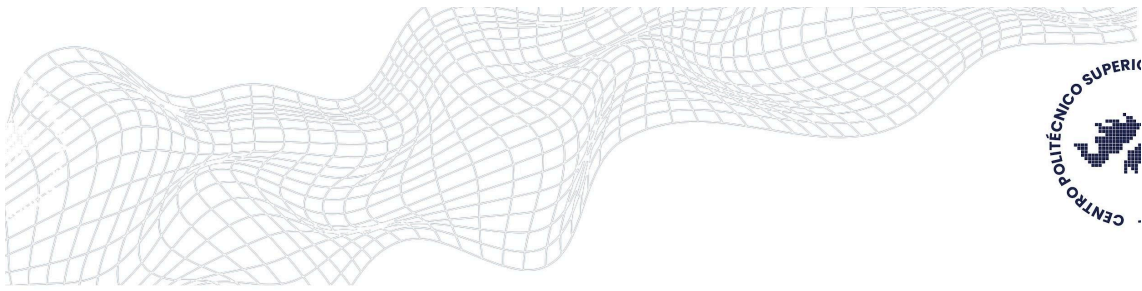
Prueba crear una línea de tiempo de los lenguajes de programación ingresando en mylens.ai y lee acerca del uso de cada uno de ellos a lo largo de la línea:



Tipos de lenguajes de programación:

Hay varios tipos de lenguajes de programación, incluyendo:

- Bajo nivel: Ensamblador, lenguajes para dispositivos embebidos. (Cerca del lenguaje de máquina)

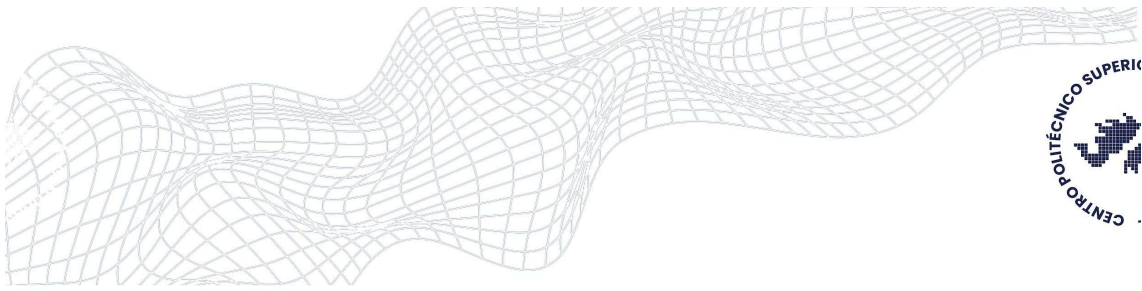


- Alto nivel: Python, Java, C++, Ruby, PHP, JavaScript. (Más fáciles de leer y escribir para humanos)
- Compilados: C, C++, FORTRAN, Ada. (Se convierten a código máquina antes de ejecutarse)
- Interpretados: Python, Ruby, JavaScript. (Se ejecutan línea por línea en tiempo de ejecución)

Otros tipos:

- Scripting: Python, Perl, Bash, PowerShell. (Automatizar tareas)
- Orientados a objetos: Java, C++, Python, Ruby. (Basados en objetos y clases)
- Funcionales: Haskell, Erlang, Lisp. (Basados en funciones matemáticas)

Ejemplos de lenguajes de programación funcionales incluyen Haskell, Lisp, Clojure y Erlang.



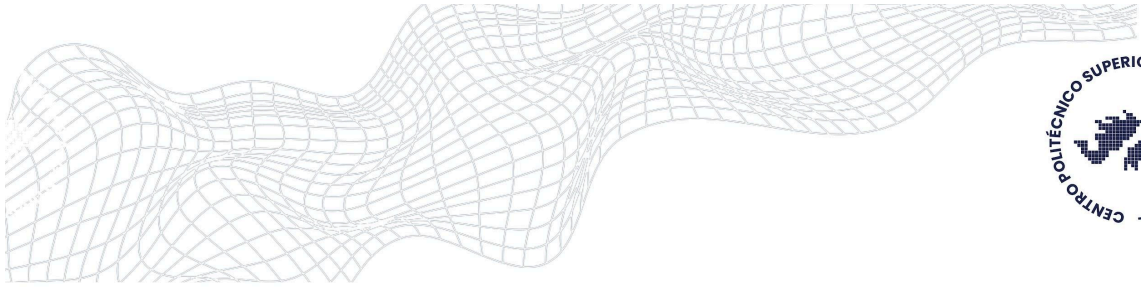
Intérpretes y compiladores:

Los lenguajes de programación pueden ser interpretados o compilados.

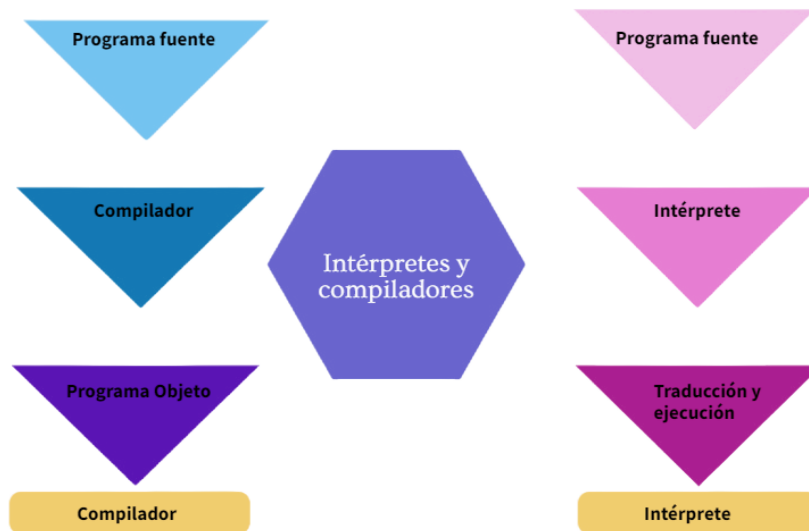
Intérpretes:

- Ejecutan el código línea por línea.
- Traducen y ejecutan cada línea al mismo tiempo.
- Más lentos que los compilados.
- Más flexibles y fáciles de usar para pruebas y depuración.

Compiladores:

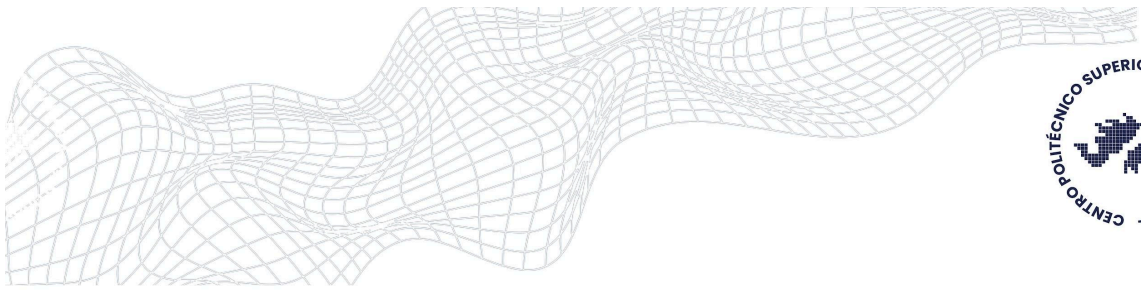


- Traducen todo el código a código de máquina antes de ejecutarlo.
- Más rápidos que los intérpretes.
- El código generado es más difícil de depurar.
- El proceso de compilación puede ser más largo.



Paradigmas de programación:

Los paradigmas de programación son diferentes enfoques o modelos que se pueden utilizar para desarrollar software. A continuación, se explican tres de los paradigmas de programación más comunes:

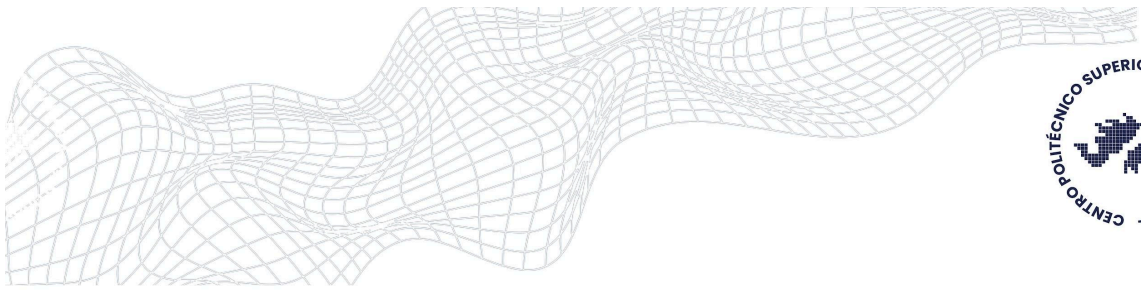


programación estructurada, programación modular y programación orientada a objetos.

Programación estructurada: Este paradigma de programación se centra en la estructura y secuencia lógica de las instrucciones de programación. Se basa en la idea de que los programas pueden dividirse en pequeñas unidades llamadas módulos, y que cada módulo tiene una función específica y claramente definida. La programación estructurada se enfoca en la simplificación y organización del código, lo que facilita su mantenimiento y mejora su legibilidad.

- Ejemplo: Un programa que calcula la factorial de un número utilizando un bucle while.

Programación modular: Este paradigma de programación es similar a la programación estructurada, pero se enfoca en la creación de módulos reutilizables que pueden ser utilizados en diferentes partes del código. Los módulos son independientes entre sí y se pueden desarrollar y probar por separado, lo que facilita el mantenimiento y la escalabilidad del código. La programación modular se basa en la idea de que la reutilización de código es fundamental para la eficiencia y la calidad del software.



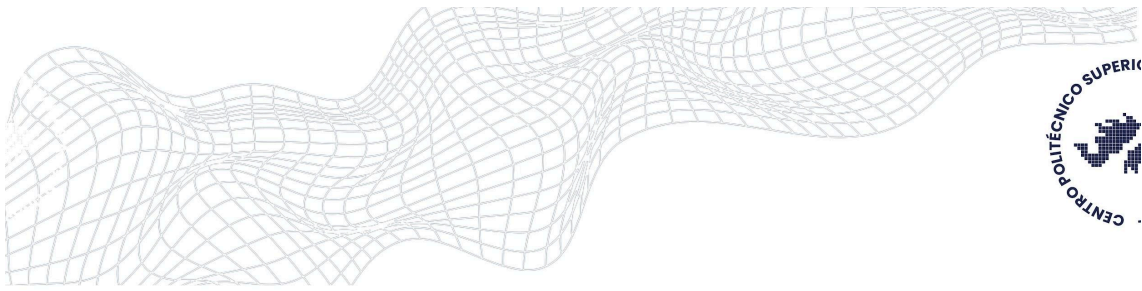
- Ejemplo: Un programa que contiene módulos separados para leer datos de entrada, procesarlos y mostrar resultados.

Programación orientada a objetos (POO): Este paradigma de programación se centra en la creación de objetos que contienen datos y funciones relacionadas. Los objetos se definen por su clase, que es una plantilla que define las características y comportamientos del objeto. La programación orientada a objetos se basa en la idea de que la programación debe reflejar el mundo real, y que los objetos pueden interactuar entre sí para realizar tareas complejas. La programación orientada a objetos se considera muy efectiva para desarrollar aplicaciones grandes y complejas, ya que facilita la organización y el modularidad del código.

- Ejemplo: Una clase "Perro" con atributos como nombre, edad y raza, y métodos como "ladrar()" y "correr()".

Los lenguajes de programación

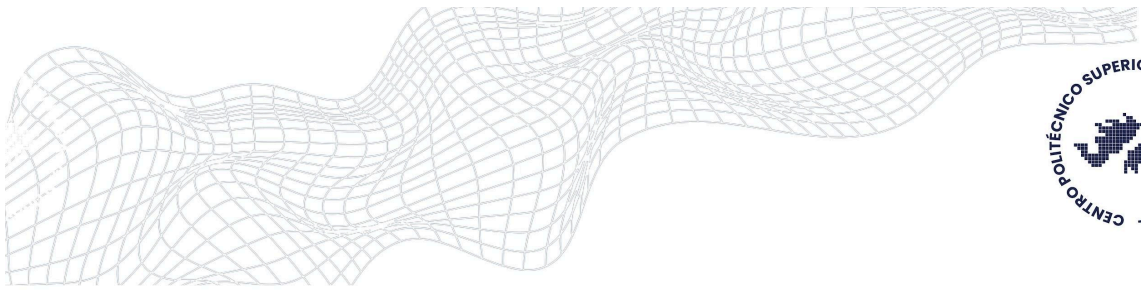
Los lenguajes de programación son una forma de comunicación entre el programador y la computadora para crear software. A lo largo de la historia, se han desarrollado muchos lenguajes de programación diferentes, cada uno con su propio conjunto de reglas y sintaxis.



Aplicaciones:

Las aplicaciones son programas de software diseñados para realizar tareas específicas en una computadora o dispositivo. Algunos ejemplos de aplicaciones incluyen procesadores de texto como Microsoft Word, hojas de cálculo como Microsoft Excel, navegadores web como Google Chrome y aplicaciones móviles como Instagram.





Historia del software:

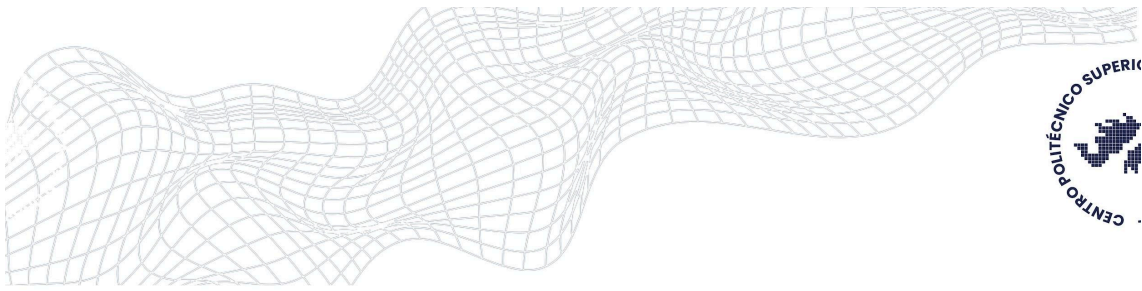
El software se ha desarrollado desde los primeros días de las computadoras. El primer programa de software fue escrito por Ada Lovelace en 1842 para la Máquina Analítica de Charles Babbage. Los primeros programas eran escritos en lenguaje de máquina, lo que hacía que la programación fuera difícil y propensa a errores. Luego se desarrollaron lenguajes de programación de alto nivel, lo que simplificó la programación y permitió a los programadores escribir programas más complejos. A medida que la tecnología avanzó, se desarrollaron nuevas aplicaciones y programas para satisfacer las necesidades de los usuarios.

La crisis del software:

La crisis del software se refiere a los problemas que surgieron en la industria del software en la década de 1960, cuando los proyectos de software se estaban retrasando y superando los presupuestos. Esto llevó a la creación de metodologías de desarrollo de software más estructuradas y a la adopción de modelos de ciclo de vida de software. Un ejemplo de la crisis del software es el Proyecto del Sistema de Control de Tráfico Aéreo de los

Estados Unidos, que se retrasó varios años y costó mucho más de lo previsto.

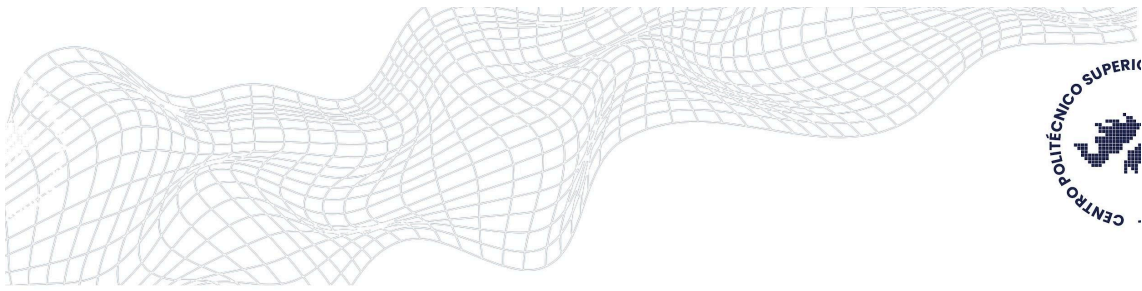
“Las Islas Malvinas, Georgias del Sur, Sándwich del Sur y los espacios marítimos e insulares correspondientes son argentinos”



El ciclo de vida de una aplicación:

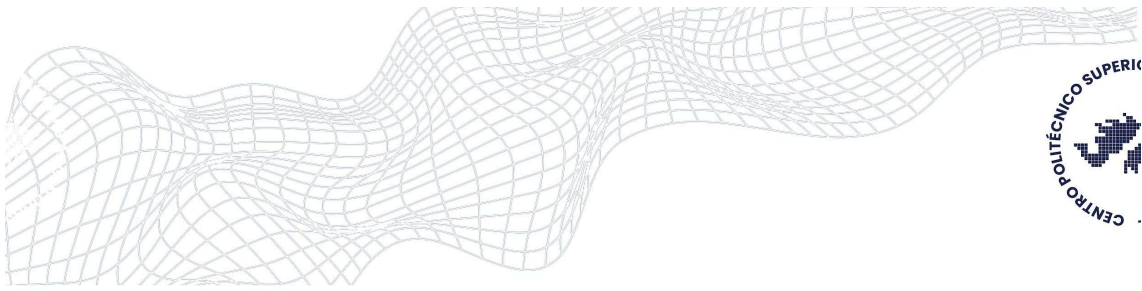
El ciclo de vida de una aplicación se refiere a las fases por las que pasa una aplicación desde su concepción hasta su retiro. Las fases típicas incluyen:

- **Análisis de requisitos:** En esta fase se identifican y documentan los requisitos de la aplicación, incluyendo las funcionalidades que debe tener y los objetivos que debe cumplir.
- **Diseño:** En esta fase se crea el diseño de la aplicación, incluyendo la arquitectura, la interfaz de usuario y los algoritmos que se utilizarán.
- **Implementación:** En esta fase se escribe el código de la aplicación utilizando el lenguaje de programación elegido.
- **Pruebas:** En esta fase se realizan pruebas para asegurarse de que la aplicación funciona correctamente y cumple con los requisitos establecidos.
- **Mantenimiento:** En esta fase se realizan actualizaciones y correcciones de errores para asegurarse de que la aplicación siga funcionando correctamente a medida que cambian los requisitos o las condiciones del entorno.
- **Retiro:** En esta fase se retira la aplicación del mercado o se reemplaza con una versión más nueva.



Te invitamos a resolver la siguiente sopa de letra para refrescar algunos de los lenguajes de programación existentes:

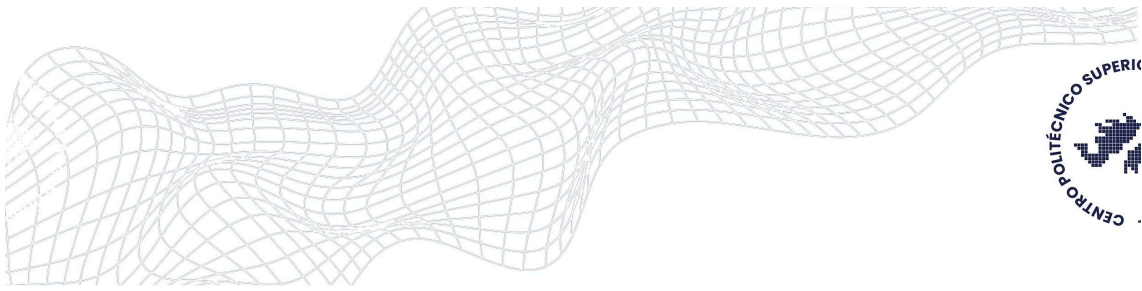
www.educima.com/wordsearches/lenguajes_de_programacin-b05aa3de02e1b3446ae7247242feb632



Errores



Los errores son comunes en la programación de software. Estos pueden ser errores de sintaxis, errores lógicos o errores de tiempo de ejecución. Por ejemplo, un error de sintaxis podría ser olvidar cerrar un paréntesis en una línea de código, lo que provocaría un error en la ejecución del programa. Un error lógico podría ser no considerar todos los posibles resultados de una operación, lo que podría dar lugar a resultados inesperados. Un error de tiempo de ejecución podría ser una división por cero, lo que provocaría una interrupción en la ejecución del programa. Es importante probar y depurar el software



cuidadosamente antes de implementarlo para minimizar la cantidad de errores.

Definición y análisis de problemas del campo informático:

Tipos de problemas:

- Rendimiento (aplicación lenta)
- Seguridad (vulnerabilidades)
- Software (errores, fallos)
- Compatibilidad (entre programas)

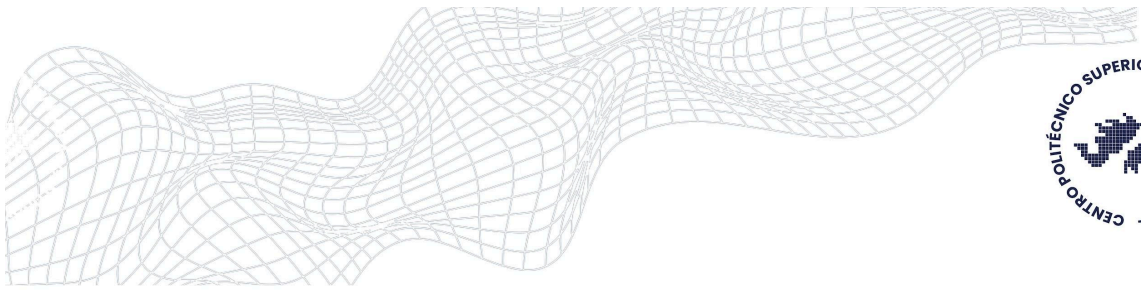
Ejemplo:

- Error de software en aplicación de procesamiento de textos.
- Pérdida de cambios en un documento por error.

Solución:

- Investigación para identificar la causa del problema.
- Desarrollo y prueba de una solución (actualización, parche).
- Pruebas adicionales para asegurar la eficacia de la solución.

Importancia:



- Definir y analizar problemas para encontrar soluciones efectivas.
- Desarrollo de software eficiente y efectivo.

La definición y análisis de problemas en el campo informático

Se refiere al proceso de identificar, comprender y resolver problemas que surgen en el desarrollo, mantenimiento y uso de sistemas informáticos. Esto implica analizar el problema en sí mismo, identificar la causa raíz y desarrollar una solución efectiva para corregir el problema.

Ejemplos:

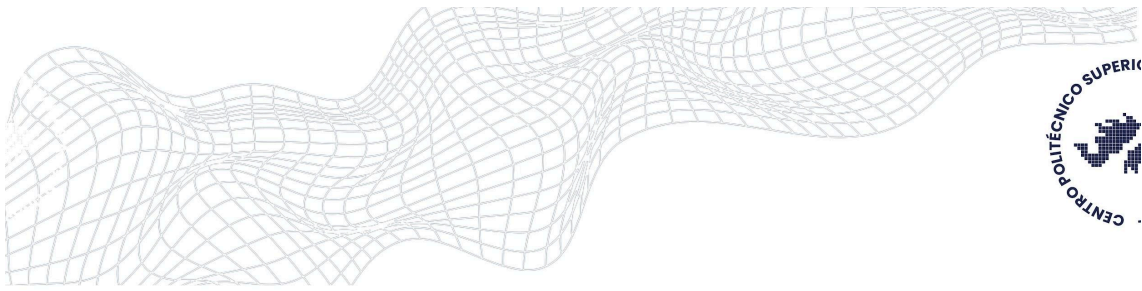
Errores de software:

- Fallas al abrir aplicaciones.
- Causa: errores de programación.
- Solución: análisis del código fuente y desarrollo de soluciones.

Problemas de hardware:

- Fallas en discos duros.
- Solución: reemplazo del hardware defectuoso.

Problemas de seguridad:



- Virus, malware, hacking.
- Solución: análisis de sistemas y comprensión de vulnerabilidades.

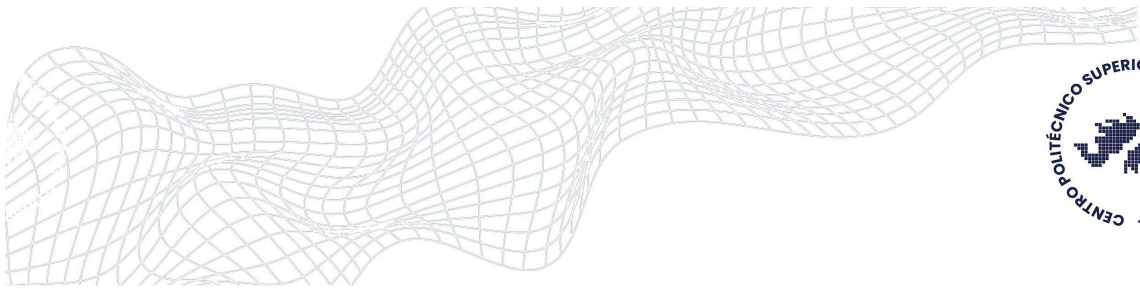
Problemas de rendimiento:

- Aplicaciones lentas o que se bloquean.
- Solución: análisis de recursos del sistema (memoria, procesador).

En resumen, la definición y análisis de problemas en el campo informático es un proceso importante para identificar y resolver problemas que pueden

surgir en el desarrollo, mantenimiento y uso de sistemas informáticos.

Ahora que has leído sobre el tema te pedimos que realices esta/s actividad/es para poner en práctica la teoría estudiada. Es importante que realices todas las actividades ya que te permitirán saber cómo vas avanzando. Si te queda alguna duda de cómo hacer esta actividad por favor consúltanos a través del foro de consultas del campus.



Actividad de cierre

1.-Trabaja el pensamiento computacional con estas actividades 'desenchufadas'

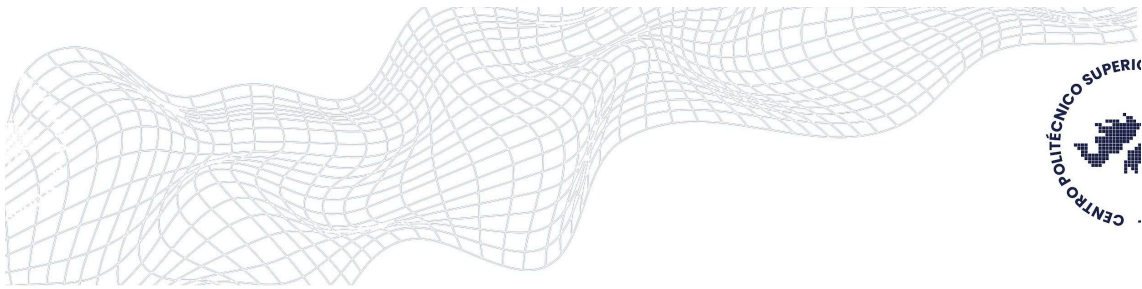
El objetivo de estas propuestas es trabajar esta disciplina reforzando destrezas como el pensamiento crítico y la resolución de problemas sin la necesidad de recurrir a un ordenador.

Pensamiento computacional

El *pensamiento computacional* es una forma de abordar y resolver problemas que involucra habilidades y procesos similares a los utilizados por las computadoras. Se trata de un *enfoque analítico y lógico* para resolver dificultades de manera estructurada.

El pensamiento computacional se basa en cuatro conceptos principales:


1. *Descomposición*: Consiste en dividir un problema complejo en partes más pequeñas y manejables. De esta manera, se simplifica el problema general y se pueden abordar los componentes individuales de manera más efectiva.

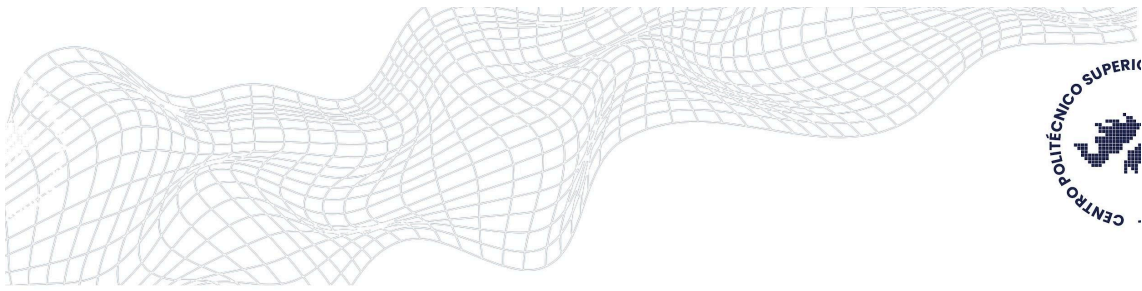


2. *Reconocimiento de patrones*: Implica identificar similitudes y regularidades en la información. Al encontrar patrones, es posible desarrollar soluciones más eficientes y generalizables.
3. *Abstracción*: Se refiere a la capacidad de filtrar información no relevante y centrarse en los aspectos esenciales de un problema. Al ignorar los detalles innecesarios, se pueden crear modelos y algoritmos más generales y aplicables a diferentes situaciones.
4. *Diseño de algoritmos*: Consiste en desarrollar una secuencia de pasos lógicos y precisos para resolver un problema. Un algoritmo es una serie de instrucciones que se pueden seguir para alcanzar un resultado deseado.

El pensamiento computacional es una habilidad que fomenta el *razonamiento crítico, la resolución de problemas y la creatividad*, ayudándonos a enfrentar desafíos de manera más eficiente.

Divide y vencerás

<p style="text-align: center;">Lavarse las manos</p> 	<ol style="list-style-type: none"> 1. _____ 2. _____ 3. _____ 4. _____ 5. _____ 6. _____
---	--



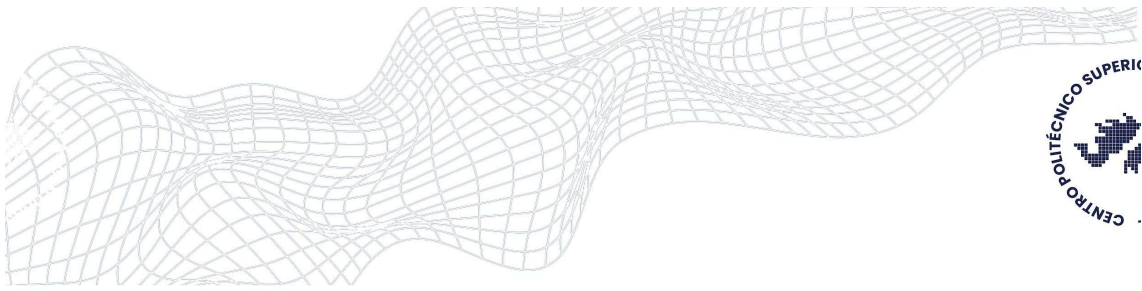
“Hoy vamos a pensar en cómo hacemos algunas de las tareas que realizamos todos los días”. Esta es la premisa sobre la que parte este ejercicio, que tiene como objetivo trabajar la descomposición y la abstracción. Para ello, el alumnado debe pensar en una tarea determinada de su día a día como, por ejemplo, lavarse los dientes, e ir descomponiéndose en pequeñas tareas: coger el cepillo, coger la pasta... y todas ellas apuntarlas en una ficha para, después, comentarlas en grupo.

Cierre

Conclusión:

En esta clase, hemos aprendido sobre los pilares de la programación y la informática. Conocimos diferentes paradigmas y lenguajes de programación, entendimos la importancia del ciclo de vida de una aplicación y cómo manejar errores. Estos conocimientos nos preparan para seguir explorando el emocionante mundo de la informática y el desarrollo de software.



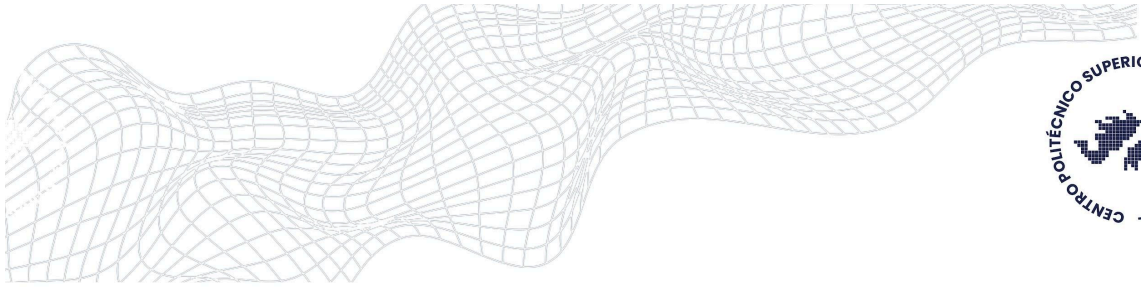


Bibliografía Obligatoria:

- FUNDAMENTOS DE PROGRAMACIÓN. Algoritmos, estructura de datos y objetos-Cuarta edición - Luis Joyanes Aguilar-McGrawHill - 2008 - ISBN 978-84-481-6111-8

Bibliografía sugerida de la Unidad:

- "Software Engineering: A Practitioner's Approach" de Roger Pressman - Este libro es un clásico en el campo de la ingeniería de software y cubre todos los aspectos del ciclo de vida del desarrollo de software.
- "Code Complete" de Steve McConnell - Este libro es una guía práctica para la escritura de código de software de alta calidad y cubre temas como la gestión de proyectos, el diseño de software y las pruebas.
- "The Mythical Man-Month" de Frederick Brooks - Este libro es considerado uno de los clásicos en la historia de la ingeniería de software y aborda temas como la gestión de proyectos, la organización del equipo y la productividad.
- "Clean Code: A Handbook of Agile Software Craftsmanship" de Robert C. Martin - Este libro se centra en la escritura de código limpio y de alta calidad y proporciona consejos prácticos para mejorar el diseño del código y la facilidad de mantenimiento.



- “The Art of Computer Programming” de Donald Knuth – Este libro es una obra de referencia en el campo de las ciencias de la computación y cubre una amplia gama de temas, desde algoritmos y estructuras de datos hasta lenguajes de programación y compiladores.