manuel.molina.rodriguez.alu@iesfernandoaguilar.es

¿QUÉ ES UN **PROGRAMA**?

Es un conjunto ordenado de instrucciones que se dan a la computadora indicando las operaciones que realizar.

¿QUÉ ES UN **SOFTWARE**?

El software es un conjunto entre programas, datos y documentación,

Programas: Son instrucciones que cuando se ejecutan proporcionan las características, función y desempeño buscados.

Datos: Incluye los datos necesarios para manejar y probar los programas y estructuras requeridas.

Documentos: Este componente describe la operación y uso de programas.

LENGUAJE DE PROGRAMACIÓN

Lenguaje artificial diseñado expresamente para crear algoritmos que puedan ser llevados a cabo por el ordenador.

¿QUÉ SON LOS **ALGORITMOS**?

Los algoritmos son secuencias precisas de instrucciones para resolver un problema.

No debe confundirse algoritmo con programa. Ya que un programa es la codificación de un algoritmo en un lenguaje de programación.

Un algoritmo debe ser **preciso**.

Un algoritmo debe estar **bien definido**.

Un algoritmo debe ser **finito**.

El algoritmo se parte en 3:

1. Entrada.
2. Proceso.
3. Salida.

Los algoritmos pueden ser representados simbólicamente por diagramas de flujo o en pseudocódigo (falso lenguaje).

DIAGRAMA DE FLUJO: **ESTRUCTURAS SECUENCIALES**

Algoritmo para obtener la suma de dos números cualesquiera.

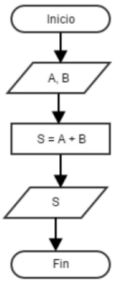
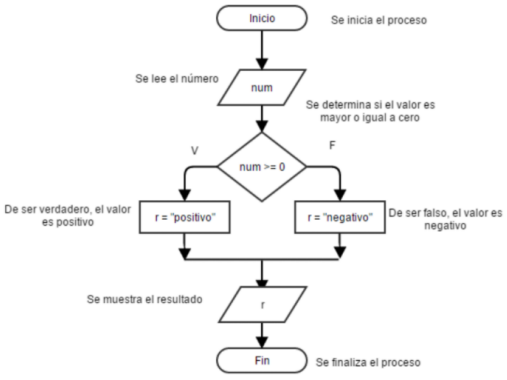
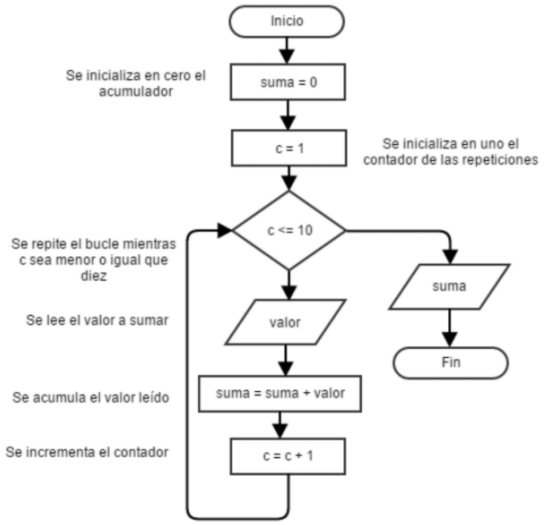


DIAGRAMA DE FLUJO: **ESTRUCTURAS SELECTIVAS**



Algoritmo para determinar si un número es positivo o negativo.

DIAGRAMA DE FLUJO: **ESTRUCTURAS REPETITIVAS**



Algoritmo para obtener la suma de diez cantidades

Construcción de un diagrama de flujo

1. **Análisis**: identificar datos de entrada y salida, grupo de pasos, puntos de toma de decisión, etc. y ordenarlos de manera cronológica.
2. **Construcción**: organizarlo gráficamente. Normalmente la organización es De arriba hacia abajo De izquierda a derecha
3. **Prueba**: se basa en hacer pruebas para ver si funciona.

ALGORITMOS. PSEUDOCÓDIGO

Es un lenguaje algorítmico que permite representar las construcciones básicas de los lenguajes de programación, pero a su vez, manteniéndose próximo al lenguaje natural.

```
INICIO
        LEER numero
        SI número < 0 ENTONCES
                absoluto = -numero
        EN OTRO CASO
                absoluto = numero
        FINSI
        ESCRIBIR absoluto
FIN
```

tienen una sintaxis muy definida, a seguir para que la computadora interprete correctamente cada orden que se le da.

La tendencia actual es hacer uso de **lenguajes de alto nivel**, están diseñados para que las personas escriban y entiendan los programas de un modo más fácil que los lenguajes máquina y ensambladores, Pero esto hace aparecer un problema, y es que los archivos de código fuente no contienen el lenguaje máquina que entenderá el ordenador. Por lo tanto, resultan incomprensibles para el procesador. Para poder generar código máquina hay que hacer un proceso de **traducción.**

Un **traductor** es un programa que recibe como entrada un texto escrito en un lenguaje de programación concreto y produce, como salida, el lenguaje máquina equivalente. El programa inicial se denomina código fuente y el programa obtenido, código objeto.
El **código objeto** a veces no es directamente ejecutable.
El proceso llamado **compilación** es la traducción del código fuente de los archivos del programa en archivos en formato binario que contienen las instrucciones en un formato que el procesador puede entender.
Por otro lado, existe un tipo de programas llamados **intérpretes**, los cuales también sirven para traducir el código fuente de un programa a código objeto, el código objeto de todos ellos deberá ser **enlazado** (unido) para obtener el deseado programa ejecutable, un programa llamado enlazador, el cual generará y guardará, en disco, un archivo ejecutable.
Hay dos tipos de **traductores**: los **compiladores** y los **intérpretes**.
Los **compiladores** son traductores que en un único proceso analizan todo el código fuente y generan el código objeto correspondiente almacenando el resultado.
Dependiendo del **compilador**, el archivo con **código objeto** generado puede ser directamente ejecutable o necesitar otros pasos previos a la ejecución.
Los **intérpretes** son traductores que hacen en forma simultánea el proceso de traducción y el de ejecución.
No crea un archivo almacenable en memoria secundaria para su posterior uso.
Los intérpretes, durante la generación de código, buscan las instrucciones en lenguaje máquina que equivalga al bloque de instrucciones fuente que se está traduciendo.
En general, a cada instrucción de alto nivel le van a corresponder varias en lenguaje máquina.

PROCESO DE OBTENCIÓN DEL CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE
Existen varias **Fases:**
Análisis lexicográfico: se leen de manera secuencial todos los caracteres de nuestro código fuente, buscando palabras reservadas, operadores, caracteres de puntuación, identificadores y organizándose todos en cadenas de caracteres denominados **lexemas.**
Análisis sintáctico-semántico: agrupa los lexemas en frases gramaticales.
Generación de código intermedio: una vez finalizado el análisis se genera una representación intermedia con el objetivo de facilitar la traducción a código objeto
Optimización de código: se revisa el código generado en el paso anterior optimizándolo para que el código resultante sea más fácil y rápido de interpretar por la máquina
Generación de código: se genera código objeto de nuestro programa en un código de lenguaje máquina relocalizable, con diversas posiciones de memoria sin establecer ya que no sabemos en qué parte de la memoria volátil se ejecutará nuestro programa.
Enlazador de librerías: se enlaza nuestro código objeto con las librerías necesarias, produciendo en último término el código final o código ejecutable.

Existen aplicaciones informáticas, llamadas **Entornos Integrados de Desarrollo (EID),** que incluyen a todos los programas necesarios para realizar todas las fases de puesta a punto de un programa, un **EID** suele proporcionar otras herramientas software muy útiles para los programadores, tales como: depuradores de código, ayuda en línea de uso del lenguaje, etc.

Un **depurador de código** permite al programador ejecutar un programa paso a paso, visualizando en pantalla qué está pasando en la memoria del ordenador en cada momento,

MÁQUINA VIRTUAL
El objetivo es que el **código intermedio**, sea común para cualquier procesador, y que el código generado a partir del intermedio sea específico para cada procesador. De hecho, la traducción del lenguaje intermedio al lenguaje máquina no se suele hacer mediante compilación sino mediante un intérprete.
La máquina virtual **Java (JVM)** es el entorno en el que se ejecutan los programas Java, su ventaja es la portabilidad de la aplicación a diferentes plataformas.
Dado que Java es un lenguaje interpretado, necesita 3 herramientas:
Un **editor de texto simple** cualquiera que sea la herramienta con la que escribimos el código.
Un **compilador del lenguaje Java**, para generar bytecode.
Un **intérprete de Java**, para poder ejecutar los programas.

CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN
Los lenguajes de programación se pueden clasificar según varios criterios.
Según la tarea a realizar, el paradigma de programación, el nivel de abstracción y la manera de ejecutarse
Según la manera de ejecutarse son 2:
Lenguajes **compilados**
Lenguajes **interpretados**

Una vez escrito el **código fuente** de un programa, este se traduce por medio de un **compilador** en un archivo ejecutable.
Un lenguaje de **programación interpretado** es aquel en el que las instrucciones se traducen o interpretan una a una, el código es traducido por el intérprete a un lenguaje entendible para la máquina paso a paso, trabaja directamente con el archivo de código fuente.

APROXIMACIÓN MIXTA
En este caso el compilador produce una representación intermedia del programa o bytecode, que después es ejecutado por un intérprete. A pesar de necesitar un proceso de compilación, estos lenguajes no se consideran realmente compilados y se siguen clasificando como interpretados.

CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN
Según el nivel de abstracción, es decir, según el grado de cercanía a la máquina:(Nivel Alto, Nivel Bajo, Código Máquina)
Lenguaje máquina: El lenguaje máquina es el único que entiende la computadora digital, es su "lenguaje natural". En él solamente se pueden utilizar dos símbolos: el cero (0) y el uno (1). Por ello, al lenguaje máquina también se le denomina lenguaje binario.

CLASIFICACIÓN. NIVEL DE ABSTRACCIÓN
Los **lenguajes de bajo nivel**, también llamados **lenguajes ensambladores**, permiten al programador escribir las instrucciones de un programa usando abreviaturas del inglés.
Lenguajes de alto nivel: dada la complejidad de desarrollar estos programas se desarrollaron lenguajes de alto nivel independientes del procesador, diseñados para que las personas escriban y entiendan los programas de un modo más fácil que los lenguajes máquina y ensambladores, y por otro lado traducibles a código máquina.
Un **lenguaje de alto nivel** permite al programador escribir las instrucciones de un programa utilizando palabras o expresiones sintácticas muy similares al inglés, otra característica importante de los lenguajes de alto nivel es que, para la mayoría de las instrucciones de estos lenguajes, se necesitarían varias instrucciones en un lenguaje ensamblador para indicar lo mismo. De igual forma que, la mayoría de las instrucciones de un lenguaje ensamblador, también agrupa a varias instrucciones de un lenguaje máquina. Por otra parte, un programa escrito en un lenguaje de alto nivel tampoco se libra del inconveniente que tiene el hecho de no ser comprensible para la computadora y, por tanto, para traducir las instrucciones de un programa escrito en un lenguaje de alto nivel a instrucciones de un lenguaje máquina, hay que utilizar otro programa que, en este caso, se denomina **compilador**.
Según la manera de abordar la tarea a realizar, pueden ser:
Imperativos, en los que se centra sobre cómo el ordenador debería hacerlo. Expresa cómo debe solucionarse un problema especificando una secuencia de acciones a realizar.
Declarativos, en los que se enfoca en qué debe hacer el ordenador. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan solo se le indica al ordenador qué es lo que se desea obtener o que es lo que se está buscando y no cómo).

¿ Qué entendemos por paradigma?
Un paradigma de programación es un enfoque particular para la construcción de software, un estilo de programación que facilita la tarea de programación.
Clasificación de los lenguajes de programación atendiendo al paradigma de programación.

CLASIFICACIÓN. PARADIGMA DE PROGRAMACIÓN

Imperativo: se expresa como debe solucionarse un problema especificando una secuencia de acciones a realizar.

Paradigma estructurado: cualquier programa, por complejo y grande que sea, puede ser representado mediante tres tipos de estructuras de control: secuencia, selección, Iteración. No hay encapsulamiento de ningún tipo. Ejemplo: Basic

Paradigma procedural: el código es básicamente un conjunto de procedimientos (funciones). Un procedimiento concreto es el inicial, y nuestro programa consiste en una secuencia de llamadas a procedimientos. Hay encapsulamiento de estado de procedimiento (variables locales). Ejemplos: C y Pascal

Paradigma orientado a objetos: el código consiste en un conjunto de objetos que colaboran entre ellos. Cada objeto tiene una identidad, estado y comportamiento. Encapsulamiento a nivel objeto. Ejemplos: Smalltalk, Java, C#, C++, Ruby y Javascript.

Declarativo: La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica al ordenador que es lo que se desea obtener o que es lo que se está buscando).

Paradigma lógico: basados en lógica formal. Tiene como característica principal la aplicación de las reglas de la lógica para inferir conclusiones a partir de datos. Ejemplo: Prolog

Paradigma funcional: código basado en funciones entendiendo por función el concepto matemático de ella. La idea es que el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produzca el resultado deseado. Ejemplos: Scala, Lisp y Haskell

La idea es que cualquier programa, por complejo y grande que sea, puede ser representado mediante tres tipos de estructuras de control:
Secuencia.
Selección.
Iteración.
Por otra parte se propone modular el programa creando porciones más pequeñas de programas con tareas específicas, que se subdividen en otros subprogramas, cada vez más pequeños.
La idea es que estos subprogramas típicamente llamados funciones o procedimientos deben resolver un único objetivo o tarea.
La construcción de programas está basada en una abstracción del mundo real.
En un programa orientado a objetos, la abstracción no son procedimientos ni funciones sino los objetos.
Estos objetos son una representación directa de algo del mundo real, como un libro, una persona, un pedido, un empleado ...
Un objeto es una combinación de datos (llamadas atributos) y métodos (funciones y procedimientos) que nos permiten interactuar con él.
En este tipo de programación, por lo tanto, los programas son conjuntos de objetos que interactúan entre ellos a través de mensajes (llamadas a métodos).
Tiene como característica principal la aplicación de **las reglas de la lógica para inferir conclusiones a partir de datos.**
Un programa lógico contiene una **base de conocimiento** sobre la que se llevan a cabo consultas.
La **base de conocimiento** está formada por hechos, que representan la información del sistema expresada como relaciones entre los datos y **reglas lógicas** que permiten deducir consecuencias a partir de combinaciones entre los hechos y, en general, otras reglas.
El paradigma es ampliamente utilizado en las aplicaciones que tienen que ver con la Inteligencia Artificial, particularmente en el campo de sistemas expertos y procesamiento del lenguaje humano.

PARADIGMAS DE PROGRAMACIÓN

Algunos lenguajes de programación pueden soportar múltiples paradigmas de programación Python que soporta la orientación objetos y el paradigma procedural y en menor medida el funcional.
Por otro lado, algunos lenguajes han sido diseñados para soportar un único paradigma de programación, ese es el caso de Smalltalk que soporta únicamente la programación orientada a objetos o Haskell que solo soporta la programación funcional

CARACTERÍSTICAS DE LOS LENGUAJES MÁS DIFUNDIDOS

Existen muchos lenguajes de programación diferentes cada uno de estos lenguajes tiene una serie de particularidades que lo hacen diferente del resto.

Los lenguajes de programación más difundidos son aquellos que más se utilizan en cada uno de los diferentes ámbitos de la informática.

Python apareció en 1991.

El creador de Python, Guido Van Rossum, es científico en computación,
trabajó para durante 7 años (2005-2012) y luego se incorporó a Dropbox

EL LENGUAJE C

Es un lenguaje de programación orientado a la implementación de sistemas operativos, concretamente UNIX. Fue desarrollado entre los años 1969 y 1972 por Dennis Ritchie, es uno de los lenguajes más eficientes en cuanto a código resultante y es el lenguaje más popular para crear software de sistemas. Es un lenguaje muy versátil, ya que trabaja tanto a bajo como a alto nivel y permite un alto control sobre la máquina.

JAVA

Es un lenguaje de programación de propósito general orientado a objetos Apareció en el año 1996 originalmente desarrollado por James Gosling, de Sun Microsystems (posteriormente adquirida por Oracle
en 2010) La última versión estable es la 10.0.1 (abril de 2018) Su sintaxis deriva en gran medida de C y C++. Las aplicaciones de Java son compiladas a bytecode, que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

SOFTWARE DE SISTEMAS Y SOFTWARE DE APLICACIÓN

Una aplicación es un tipo de programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. Esto lo diferencia principalmente de otros tipos de programas (sistema) que realizan tareas más avanzadas y no pertinentes al usuario común, como los sistemas operativos (que hacen funcionar al ordenador, como Windows, Mac o Linux), las utilidades (que realizan tareas de mantenimiento o de uso general), y los lenguajes de programación (con el cual se crean los programas informáticos).

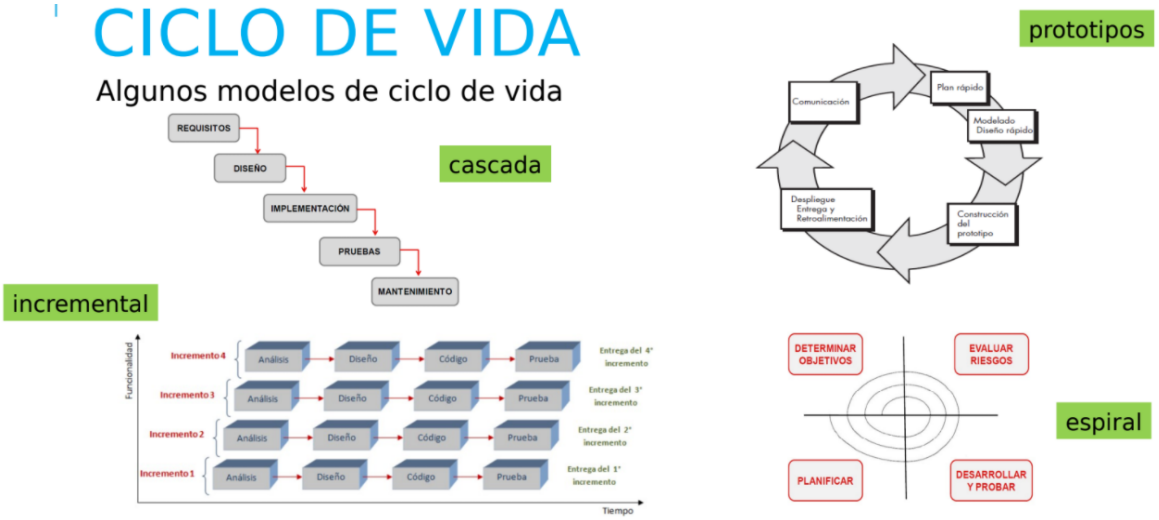
DESARROLLO DE SOFTWARE

Toda aplicación informática, ya sea utilizada en un ordenador de sobremesa o un ordenador portátil, o sea utilizada en dispositivos móviles como un teléfono móvil de última generación o una tableta, ha seguido un procedimiento planificado y desarrollado detalle por detalle para su creación. Este irá desde la concepción de la idea o de la funcionalidad que deberá satisfacer esta aplicación hasta la generación de uno o varios ficheros que permitan su ejecución exitosa.

DESARROLLO DE SOFTWARE: CICLO DE VIDA

Para convertir esta concepción de una idea abstracta en un producto acabado que sea eficaz y eficiente habrá muchos más pasos, muchas tareas que hacer. Este conjunto de etapas en el desarrollo de software corresponde al concepto de ciclo de vida del software. No en todos los programas ni en todas las ocasiones el proceso de desarrollo llevará fielmente las mismas etapas; no obstante son unas directrices muy recomendadas.

Análisis, Diseño, Codificación, Implantación, Mantenimiento



Análisis (qué se quiere) Define los requisitos del software que hay que desarrollar. Inicialmente esta etapa comienza con una entrevista con el cliente, que establecerá lo que quiere o lo que necesita.
Diseño (cómo se va a hacer) uno de los objetivos principales de esta etapa es establecer las consideraciones de recursos del sistema, tanto físicos como lógicos. Se define por tanto el entorno que requerirá el sistema. Y se crean diagramas que modelan el funcionamiento del mismo.
Codificación el software que implemente todo lo analizado y diseñado anteriormente en un lenguaje de programación concreto, haciéndolo de una forma lo más modular posible para facilitar el posterior mantenimiento o manipulación por parte de otros programadores. Se traduce el análisis y diseño en una forma legible por la máquina
Pruebas con una doble funcionalidad las pruebas buscan confirmar que la codificación ha sido exitosa y el software no contiene errores a la vez que se comprueba que el software hace lo que debe hacer, que no es necesariamente lo mismo.
Implantación del software en el entorno donde los usuarios finales lo utilizarán. Cabe destacar que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de ambas aplicaciones durante un proceso de adaptación.
Mantenimiento son muy escasas las ocasiones en las que un software no vaya a necesitar de un mantenimiento continuado. En esta fase de desarrollo de software se arreglan los fallos o errores o se agregan nuevas funcionalidades que suceden cuando el programa ha sido implementado.

DESARROLLO DE SOFTWARE: METODOLOGÍA

Un objetivo de décadas en el desarrollo de software ha sido encontrar procesos y metodologías que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del software
Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo de software.
las metodologías tradicionales que se centran especialmente en el control del proceso, estableciendo las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán.
Como puede ser el caso de Métrica v.3 que es una metodología de planificación, desarrollo y mantenimiento de sistemas de información desarrollada por el Gobierno de España que cubre tanto desarrollos estructurados como orientados a objetos.

METRICA V3.0

Desarrollada por el Ministerio de Administraciones Públicas. Se trata de una metodología para la planificación, desarrollo y mantenimiento de los sistemas de información de una organización.
Planificación de Sistemas de Información (PSI)
Desarrollo de Sistemas de Información (DSI)
Mantenimiento de Sistemas de Información (MSI)
Desarrollo de sistemas de Información

1. Estudio de viabilidad del sistema (EVS)
2. Análisis del sistema de información (ASI)
3. Diseño del sistema de información (DSI)
4. Construcción del sistema de información (CSI)
5. Implantación y aceptación del sistema (IAS)

DESARROLLO DE SOFTWARE: METODOLOGÍA

A partir del 2001, surge un punto de inflexión, con el surgimiento de otro tipo de metodologías conocidas como las metodologías ágiles para el desarrollo de software.
Las metodologías ágiles cuales dan mayor valor a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos pequeños/medianos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

DESARROLLO DE SOFTWARE: ROLES

Los roles no son necesariamente rígidos y es habitual que participen en varias fases del proceso de desarrollo:
Jefe de proyecto: dirige todo el proyecto de desarrollo de software. Se encarga de entregar un producto de calidad, manteniendo el coste dentro de las limitaciones del presupuesto del cliente y entregar el proyecto a tiempo.
Analista funcional: participa el la etapa de análisis. Se encarga del análisis de requisitos del software a construir.
Diseñador de software: Participa de la etapa de diseño. Diseña la solución a desarrollar a partir de los requisitos.
Arquitecto: conoce e investiga frameworks, y tecnologías, revisando que todo el proceso se lleva a cabo de la mejor forma y con los recursos más apropiados.

Analista programador: comúnmente llamado desarrollador, domina una visión más amplia de la programación. Participa en la etapa de diseño y codificación
Programador: escribe código fuente del software. Participa solo en la codificación del software