**TAREA 11**

**Turing**

Alan Turing, considerado uno de los primeros precursores de las ciencias de la computación, la informática y padre de la Inteligencia Artificial.

Durante la segunda guerra mundial, fue capaz de descifrar, los mensajes encriptados del ejército alemán

**Programación**

Organizar una secuencia de acciones que ejecutados de forma ordenada consiguen, resolver un problema, realizar una tarea,…, es decir un objetivo

**inteligencia-artificial**

Campo de la ciencia cuyo objetivo es hacer que las maquinas puedan imitar comportamientos inteligentes

Otros

**Ciencia computacional** =>

Campo de estudio, relacionado con la construcción de modelos matemáticos y técnicas numéricas para resolver problemas, científicos, de ciencias sociales y de ingeniería

**Informática** => Procesamiento Automático de la información, mediante dispositivos electrónicos y sistemas computacionales.

Un sistema informático debe cumplir 3 tareas básicas:

* Entrada => Captación de la información
* Procesamiento
* Salida => transmisión de resultados

Estas 3 tareas en conjunto se conocen como algoritmo

**Arquitectura de Von Neumann**

Arquitectura actual de los ordenadores:

* CPU
* Memoria central
* Dispositivos de E/S

**Maquina universal de Turing**

Compuesta de :

1. Una cinta, tan larga como se necesite y dividida en casillas => Será la MEMORIA

En esta podemos escribir símbolos, por ejemplo, ceros y unos

En nuestros ordenadores actuales, la cinta es la MEMORIA

1. Una cabeza, capaz de moverse por la cinta (a izqda. y drcha.) y que será capaz de LEER y ESCRIBIR, símbolos en la cinta anterior

En nuestros ordenadores el microprocesador, simula la cabeza y le encarga de leer y ejecutar los programas

1. Un programa, que le diga a la cabeza que es o que tiene que hacer.

Este programa puede estar en la cinta, codificado mediante ceros y unos

**Algoritmo**

Conjunto ordenado y finito de operaciones ordenadas, que ejecutadas permiten dar solución a un problema

Características básicas de un algoritmo:

* Tiene principio y fin
* Funciona en secuencia, son pasos ordenados
* Las secuencias son concretas, no dejan lugar a la ambigüedad
* Son abstractos, modelos o guías para ordenas procesos

**TAREA 12**

**Transistor** =>

Componente integrado en los circuitos electrónicos.

Pueden amplificar señales, pero en informática son los encargados de dejar pasar (bit a 1) o no (bit a 0) la corriente.

Componente electrónico semiconductor (dependiendo de factores se comporta como aislante o como conductor) que puede modificar una señal eléctrica de salida, en respuesta a una señal de entrada, "emite", una señal de salida

Juntando transistores se forman las puertas lógicas

**ley-de-moore** =>

Acuñada por Gordon Moore, cofundador de Intel

La ley de Moore dice que cada 2 años, se duplica el número de transistores en un microprocesador.

Según esta ley, inicialmente, la complejidad de los circuitos integrados (velocidad del procesador, nº de transistores por pulgada) se duplicaría cada año y disminuiría su costo. Mas tarde ajusto su ley para decir que este reajuste se haría cada 2 años y no cada año.

**puertas-lógicas** =>

Dispositivos electrónicos, capaces de realizar operaciones lógicas básica

Son las piezas con las que se construyen los circuitos.

Sumando muchas puertas lógicas se pueden realizar cálculos muy complejos

OTROS

Algebra booleana =>

Rama del algebra utilizado en electrónica digital (George Boole 1854).

Es un método para simplificar los circuitos lógicos (0 y 1) en electrónica digital

electrónica digital =>

Está basada en cero y unos trabaja entre 0 y 10 voltios (hay otros rangos dentro de este)

**TAREA 13**

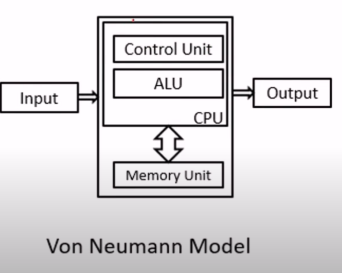
**Periférico** =>

Dispositivo hardware externo que se conecta a la CPU y que permite la entrada o salida de datos/información al ordenador

Clasificación entrada, salida, entrada/ salida y almacenamiento

**arquitectura-von-neumann** =>

Arquitectura de ordenadores, basada en los siguientes componentes

* CPU
* Memoria central (almacena tanto datos como instrucciones(programas))
* Dispositivos de E/S
* 

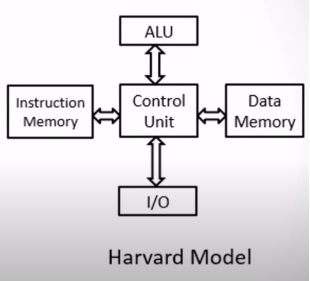
En esta arquitectura la CPU puede estar o bien leyendo una instrucción o leyendo/escribiendo datos desde/hacia la memoria, pero ambos procesos no pueden estar ejecutándose al mismo tiempo ya que las instrucciones y los datos utilizan el mismo sistema de buses

**arquitectura-harvard** =>

Separa las memorias para contener los datos por un lado y los programas por otro

También separa las ALU y la CI

Puede ejecutar las instrucciones en un único ciclo de reloj



Con esta esta arquitectura se puede tanto leer una instrucción como realizar un acceso a la memoria de datos al mismo tiempo, ya que ambas memorias están separadas y cada una tiene su propio sistema de buses

**sistema-operativo** =>

Es el programa (software), más importante de un ordenador. Es el intermediario entre el usuario y el hardware, nos permite darle ordenes al ordenador para que este haga lo que necesitamos. Sin el SO el ordenador es una máquina inútil

Gestiona los recursos hardware de un ordenador y provee servicios a los programas de software, para que puedan ejecutarse de modo privilegiado respecto al resto de programas en ejecución.

**Software** =>

Programas que hacen posible el funcionamiento de un ordenador. El software le indica al hardware que hacer y cómo hacerlo.

Principales tipos de software:

* Software de sistema.
  + Programa/s que permiten al usuario relacionarse con el sistema, pe, sistemas operativos
* Software de programación:
  + Programas que permiten a un programador el desarrollo de sus propios programas informáticos, pe, compiladores editores, IDEs,..
* Software de aplicación
  + Programas diseñados para facilitar a los usuarios la realización de determinadas tareas. Pe, video juegos, aplicaciones de ofimática, navegadores…

**Hardware** =>

Elementos físicos que componen un ordenador. Es el canal físico que permite que las instrucciones dadas por el software puedan realizarse

**raspberry pi** =>

Mini ordenador construido sobre una placa reducida o placa única (SBC), que sirve para construir proyectos electrónicas, aprender programación o realizar funciones habituales de cualquier PC, navegar por internet, ejecutar apps de escritorio (hojas de cálculo,…)

**Arduino** =>

Plataforma abierta compuesta por hardware y software, para el desarrollo fácil de proyectos electrónicos

Su parte hardware la placa base es un microcontrolador, que es un circuito integrado capaz de ejecutar las ordenes grabadas en su memoria

**TAREA 14**

**Compilador** =>

Traducen al completo el programa que se va a ejecutar escrito en su lenguaje al código máquina del sistema, se guarda esta traducción y se ejecuta, con lo que esta traducción de puede almacenar para ser usada posteriormente sin necesidad de volver a traducir.

Se compone de las siguientes fases:

* Fase de análisis, compuesta a su vez por
  + *Analizador léxico (Scanner)*

Primera fase, divide el programa en tokens. Cada token es clasificado según su resultado

* *Analizador sintáctico (Parse)*

Segunda fase , genera un árbol sintáctico.

* *Analizador Semántico*

Ultima fase, analiza todo el programa y valida que todo tenga un significado exacto y nada pueda fallar en tiempo de ejecución.

* Fase de síntesis

En esta fase se genera el código intermedio

* Optimización del código

A veces se incluye, como parte de la fase anterior.

Su objetivo es mejorar el código intermedio, para que el código máquina sea realmente muy radio de ejecutar.

**Intérprete** =>

Programa informático capaz de analizar y ejecutar otros programas.

Los intérpretes solo realizan la traducción a medida que se necesita y no guardan el resultado de dicha traducción.

Los programas interpretados suelen ser más lentos que los ejecutados, ya que se deben ir traduciendo mientras se están ejecutando, pero son más flexibles desde el punto de vista de la programación y su depuración.

No dependen de la máquina donde se ejecuta sino del propio interprete o máquina virtual

**Lenguaje de programación** =>

Sistema estructurado de comunicación, formado por:

* Conjunto de símbolos
* Palabras clave
* Reglas sintácticas y semánticas.

Permite el entendimiento entre un programador y una máquina

**Editor de código** =>

Un editor de código es un editor de texto diseñado específicamente para editar el código fuente de programas informáticos

Características de un buen programa para editar código

* No debe tener formato (negrita cursiva) Word por ejemplo descartado
* Debe tener resaltado de sintaxis, al escribir código el programa reconoce el lenguaje y colorea ciertas palabras (palabras clave) que te ayudan a leer/interpretar el código, por ejemplo aquí descartaríamos un block de notas
* Debe tener autocompletado, ofrece sugerencias de lo que quieres escribir
* Debe detectar errores de sintaxis (para funciones o palabras reservada)

[Editor vs IDE ¿Qué usar para programar?](https://www.youtube.com/watch?v=dx_sfmbV-bo)

Editor de código,

Es un programa ligero (no necesitas grandes recursos del ordenador, no necesita mucha RAM, no te pide mucho procesador), abre archivos rápidamente.

Pero no por ello es simple, ya que es habitual que se le puedan instalar plugins para irle dando nuevas capacidades y potencias, eso si, en el caso de los editores el usuario manualmente es el que instala y configura cada plugin que quiera instalar manualmente

Puede soportar (autocompletado, resaltado de sintaxis, ayudas, detección de errores…) múltiples lenguajes y si no instalarle el plugin correspondiente para que soporte ese lenguaje

Orientado a ficheros



IDE "**E**ntorno de **D**esarrollo **Integrado"**

Es como si tuvieses un editor de texto, donde te has tomado mucho tiempo instalando y configurando plugis, por lo que puedes crear un proyecto de inicio a fin, sin perder el tiempo en configurar

Enfocado a proyectos y no a archivos

Es más pesado que un editor, es decir solicitan bastante uso de recursos de máquina, procesador, RAM

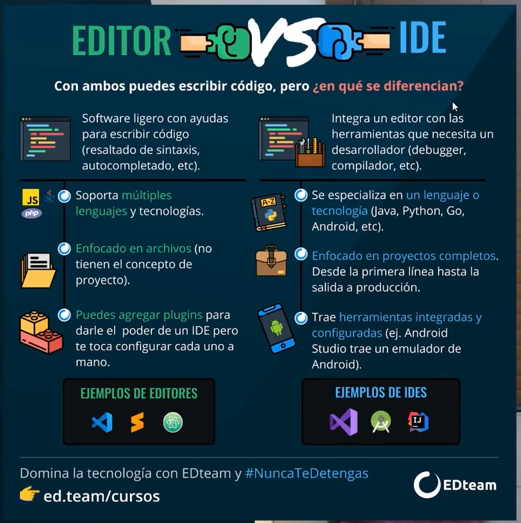
Tiene compiladores integrados, emuladores, terminales, se une con Git

Refactorización, se trata de cambiar los nombres de variables clases o funciones a lo largo del todo el proyecto

Como todo el proyecto está indexado,

Los IDE están pensados para lenguajes específicos, por ejemplo, Spyder para Python, Eclipse para Java, Android Studio para crear proyectos android

IDE vs Editor



**Lenguaje de alto nivel** =>

El LP de alto nivel, es el que se aproxima más al lenguaje humano que al lenguaje binario de las computadoras.

Permite al programador escribir instrucciones del programa utilizando expresiones similares al inglés, por lo que son más fácilmente legibles para las personas que para las maquinas, los ordenadores.

Normalmente una instrucción de alto nivel necesita de varias instrucciones de código máquina para indicar los mismo.

**Lenguaje de máquina (código máquina)** =>

Muy utilizado ya que es el que va a comunicar las instrucciones a la máquina. Su uso es indispensable, ya que es el que comunica las instrucciones a la máquina.

Es el sistema de códigos directamente interpretable por la máquina. Estos códigos determinan las acciones que han de ejecutarse por la máquina.

Es específico para cada máquina, aunque pueden ser similares.

**TAREA 15**

**Lenguaje de programación** =>

Referencia incluida en el punto anterior

**Python** =>

Lenguaje de programación interpretado y de sintaxis muy sencilla.

Es uno de los lenguajes más utilizados.

Se puede utilizar tanto para desarrollo web como para escritorio y se suele utilizar mucho en ciencia de datos (Big Data, machine Learning, …) y seguridad informática

**R** =>

Lenguaje de programación interpretado, muy utilizado para el análisis estadístico y gráfico.

Es uno de los lenguajes de programación más utilizados, para la ciencia de datos.

Gratuito y de código abierto

**Lisp** => (**Lis**t **P**rocessing, procesamiento de Listas). El elemento fundamental en LISP es la lista, a que tanto los programas como los datos son listas. Es un lenguaje de programación de alto nivel, y fue el primer lenguaje de IA

**Java** =>

Se trata de un lenguaje de alto nivel, que algunos clasifican como "parcialmente compilado".

Cualquier desarrollo en java debe ser compilado lo que puede dar lugar a un .class o un .jar (si contiene muchas clases)

Que es un compilado a bytecode (formato binario que no es entendido directamente por ninguna arquitectura hardware, pero si por la JVM) y que después se ejecuta sobre su máquina virtual (JVM), lo que hace que se pueda ejecutar sobre diferentes SO, sin que tenga que ser compilado específicamente para cada uno de ellos.

**Lenguaje de alto nivel** =>

Referencia incluida en el punto anterior.

Son lenguajes que como seres humanos podemos interpretar

**Lenguaje de bajo nivel** =>

Lenguaje de bajo nivel, que interactúa directamente con el hardware.

Lenguaje máquina

**Lenguaje interpretado** =>

Es un lenguaje que se convierte en lenguaje maquina en tiempo de ejecución, es decir, cuando el programa se está ejecutando.

**Lenguaje compilado** =>

Se traduce primero a lenguaje máquina y esta traducción es la que se ejecuta en el ordenador.

**Compilador** =>

Es un traductor que traduce un programa de un lenguaje de programación a otro.

Normalmente traducen a lenguaje máquina

**TAREA 16**

**Repositorio** =>

Espacio centralizado que almacena y organiza, contenidos diversos

**Control de versiones** =>

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que sea posible recuperar versiones especificas más adelante.

**Github** =>

Plataforma en la nube, que permite alojar proyectos, utilizando la tecnología Git, como sistema de control de versiones

**TAREA 17**

**Lenguaje de programación** =>

Referencia incluida en la tarea 14

**javascript =>**

R

**Python** =>

Referencia incluida en la tarea 15

**Php (**inicialmente **P**ersonal **H**ome **P**age**,** ahora **H**ypertext **P**reprocessor**)** =>

Lenguaje de código abierto, que aunque es de propósito general, resulta muy adecuado para desarrollo web y que puede ser incrustado en HTML.

A diferencia de JS, se ejecuta en el servidor (backend) utilizando un intérprete php y devuelve HTML que será enviado al cliente

**Java** =>

Referencia incluida en la tarea 15

**HTML** (**H**yper**T**ext **M**arkup **L**anguage) =>

Es un lenguaje de marcas para la generación de páginas WEB.

Con este lenguaje se define el contenido de las páginas WEB (párrafos, textos, imágenes botones, ….), haciendo uso de sus distintas etiquetas.

Ultima versión HTML5, que incluye

HTML => Da forma a la estructura de la pagina

CSS => Da estilos a la página (colores, tipos de letras tamaños, ..)

JavaScript => da "dinamismo" a la página web, permitiendo que se rellenen dinámicamente parte de la página en función por ejemplo a acciones previas sobre esta

**CSS (C**ascading **S**tyle**S**heets**,** Hojas de estilo en cascada**)** =>

Lenguaje para definir la apariencia de una página web. Se usa en el diseño visual de un sitio web, cuando las páginas que lo forman están generadas con HTML o XML.

**R** =>

Referencia incluida en la tarea 15

**SQL** (Structured **Q**uery **L**anguage) =>

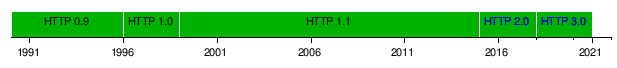
Lenguaje para la gestión de BBDD relacionales. Permite acceder y actualizar la información, los datos dentro de la BBDD de forma sencilla

.

**HTTP** (**H**yper**T**ext **T**ransfer **P**rotocol) =>

Protocolo de comunicación que permite transferencias de información en la WEB.

Al no guardar estado (conexiones anteriores), hace uso de las cookies para ello



**FTP** (**F**ile **T**ransfer **P**rotocol) =>

Protocolo de transferencia de archivos, para arquitectura cliente -servidor.

Los sistemas deben estar conectados a una red TCP/IP

**VNC** (**V**irtual **N**etwork **C**omputing) =>

Programa (basado en la restructura cliente-servidor) que permite observar las acciones del ordenador servidor, remotamente a través de un ordenador cliente

**VPN** (**V**irtual **P**rivate **N**etwork) =>

Es una tecnología de red de ordenadores, que permita la extensión segura de una LAN (red de area local) de forma segura, sobre una red pública o no controlada.

**TAREA 21**

**Algoritmo** =>

Debe de haber un INPUT (entrada de datos)

Seguimos una SECUENCIA DE PASOS, ordenada y bien definida

Genera un resultado, un OUTPUT => datos de salida

Conjunto de pasos finitos y bien organizados que permite la resolución de un problema o el desarrollo de una tarea.

Sus partes fundamentales son 3:

ENTRADA => Debe de haber un INPUT (entrada de datos)

PROCESO => Seguimos una SECUENCIA DE PASOS, ordenada y bien definida

SALIDA => Genera un resultado, un OUTPUT => resultado que se genera al aplicar los pasos del proceso sobre los datos de entrada

Un algoritmo nos permite llevar a cabo una tarea o encontrar la solución a un determinado problema a través de:

* una serie de instrucciones bien definidas y estructuradas, que además
* deben estar en orden, es decir una tras otra, y
* ser finitas, es decir tener una solución posible.

De este modo, definidos:

* un estado inicial y una entrada,
* si seguimos las instrucciones formuladas
* llegaremos al estado final y encontraremos la solución al problema o alcanzaremos el objetivo definido.

**Diagrama-de-flujo** =>

Un diagrama de flujo es una representación gráfica, que describe un proceso, sistema o algoritmo informático.

Un diagrama de flujo se utiliza para:

* documentar,
* estudiar,
* planificar,
* mejorar y
* comunicar

procesos, que suelen ser complejos en diagramas claros y fáciles de comprender

**TAREA 22**

**Ada Lovelace** =>

Matemática y escritora inglesa del siglo XIX.

Considerada por muchos, como la primera **programadora** de la historia.

Introdujo el concepto de **bucle** o subrutina, además de reutilizar la idea que vio en su visita a unos telares, de utilizar las **tarjetas perforadas** como medio de introducir código en las máquinas para poder programarla.

Introdujo también el **concepto de ordenador**, al ver que la máquina analítica, podría procesar más allá de cálculos matemáticos.

También llego a pensar en la posibilidad de que las máquinas pensaran por sí mismas (**IA**), aunque finalmente desechó esta idea.

**Bucles** =>

Conjunto de instrucción contenidas en un programa, que se repite un número determinado de veces mientras/hasta que se cumple una condición.

**Lenguaje de Programación** =>

[Definición](#lenguajeDeProgramacion) hecha en la tarea 24

**TAREA 23**

**Algoritmo de cifrado simétrico** =>

Es un algoritmo de cifrado, que utiliza la misma clave, para **cifrar y descifrar** el mensaje, por lo que ambas partes (emisor y receptor) deben conocer dicha clave. Este tipo de criptografía es conocida también, como criptografía de clave secreta o criptografía de una clave.

Ejemplo de este tipo de criptografía es la maquina **Enigma** utilizada por el ejército alemán durante la 2ª guerra mundial.

Sus limitaciones están, en asegurarse de tener un canal seguro para la transmisión de claves y que esta no llegue a otros receptores no autorizados y que, si se necesitan un gran número de claves, puede ser difícil de gestionar.

Este tipo de cifrado aporta **confidencialidad**

**Algoritmo de cifrado asimétrico** =>

Este tipo de algoritmos de cifrado utilizan dos claves, una pública y otra privada.

Estos algoritmos se basan en funciones matemáticas, fáciles de resolver en un sentido, pero si no se conoce la clave, difíciles de resolver en el sentido inverso.

Este tipo de cifrado aporta **autenticidad, integridad y no repudio**

**Funciones en programación** =>

Conjunto de instrucciones o sección dentro de un programa, que calcula un valor de manera independiente al resto del programa.

Su estructura básica está compuesta por:

* + Parámetros o valores de entrada a la función
  + El código de la función (operaciones que debe realizar)
  + Resultado, o valor final que devolverá la función cuando termine.

Conceptualmente se diferencia del procedimiento en que este último, solo ejecuta una tarea, pero no devuelve ningún valor.

Ventajas, se puede invocar este “trozo” de código cuando se necesite, sin tener que rescribirlo y facilita el mantenimiento del código, ya que, en caso de necesitar ajustes, solo se deberá hacer una vez.

**TAREA 24**

**Convertidor analógico digital** =>

Dispositivo que convierte una entrada analógica a un número digital.

Su salida, puede expresarse en diferentes codificaciones, como la binaria.

Las señales analógicas, emplean magnitudes con valores continuos

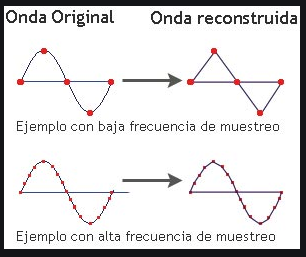
Las señales digitales, utilizan valores discretos.

Estos convertidores, siguen 3 pasos en su proceso conversor:

* 1. Muestro: Toma de muestras de la señal a intervalos periódicos
  2. Cuantificación: Se encarga de convertir las muestras tomadas de la señal analógica a una sucesión de valores discretos
  3. Codificación: Representación de los valores discretos de la muestra en el código elegido (aquí binario)

**Frecuencia de muestreo** =>

La frecuencia de muestreo o tase de muestreo, es el número de muestras por unidad de tiempo que se toman de una señal continua, para producir una señal discreta.



http://musiki.org.ar/Frecuencia\_de\_muestreo

**Transistores bipolares** =>

Dispositivo, que puede tener 2 funciones:

* + Amplificar la señal/tensión
  + Interrumpir la señal/tensión

Nacen de la necesidad de controlar el flujo de la corriente eléctrica

Funcionan como una llave de paso y se componen de 3 terminales:

* + Emisor
  + Colector
  + Base, el da tensión y hace que pase o no la señal

**Sistema binario** =>

El sistema binario, es un sistema de numeración de base 2 y que por tanto utiliza únicamente dos símbolos (el cero y el uno) y donde los números se representan mediante una combinación de estos.

**Algebra booleana =>**

Es un sistema matemático, que se utiliza para representar cualquier circuito lógico en forma de ecuaciones algebraicas. El algebra de Boole, es el algebra de 2 valores, 0 y1, Verdadero y Falso. Básicamente es un lenguaje en modulo 2.

Rama del algebra utilizado en electrónica digital (George Boole, matemático 1854). Es un método para simplificar los circuitos lógicos (0 y 1) en electrónica digital

 electrónica digital =>

Está basada en cero y unos trabaja entre 0 y 10 voltios (hay otros rangos dentro de este)

**TAREA 25**

**Arquitectura-cliente-servidor** =>

Modelo de desarrollo de aplicaciones, donde las transacciones se dividen en procesos independientes que cooperan entre sí, para el intercambio de información, servicios o recursos.

Si tenemos en cuenta, la potencia de cálculo que se ejecuta en cada parte podemos clasificarlos como ligeros y pesados. El conjunto suele ser:

* + Cliente ligero / Servidor Pesado
  + Cliente Pesado / Servidor Ligero

Cliente ligero / Servidor Pesado:

La potencia de cálculo está en el servidor y suelen ser servidores dedicados (con tareas específicas) servidores de DNS, web, de BBDDs.

Aquí se requiere gran bando de ancha por el gran número de transacciones entre C/S

Cliente Pesado / Servidor Ligero:

La potencia de calculo está en la parte del cliente y el servidor suele dedicarse principalmente al almacenamiento de datos

No requieren tanto de la red, pero su información está distribuida, ejemplo, cajeros automáticos.

**FTP** =>

[Definido anteriormente](#FTP)

**Protocolo TCP/IP** =>

Es el conjunto de protocolos que rigen internet. Permiten establecer la conexión e intercambiar datos a través de un transporte fiable.

* + ***TCP***, **T**ransmission **C**ontrol **P**rotocol => Se encarga de la entrega de datos, una vez conocida la IP.
  + **IP**, **I**nternet **P**rotocol => Encargado de obtener la dirección a la que se entregan los datos.

Está compuesto por 4 capas:

* + Capa de acceso al medio. Se encargar de la parte física del envío y recepción de datos mediante el cable Ethernet, red inalámbrica, tarjeta de red…
  + Capa de internet. Controla el movimiento de paquetes en la red.
  + Capa de transporte. Proporciona una conexión fiable entre 2 dispositivos. Los datos los divide en paquetes.
  + Capa de aplicación. Es la parte con la que el usuario interactúa, página web, correo electrónico, …

**Servidor web** =>

Es uno de los tipos de servidores dedicados que existen

Programa que utiliza el protocolo HTTP. Se trata de un protocolo para la transferencia de hipertexto o paginas HTML, en la web. Su función es devolver páginas cuando el cliente las solicita.

Aquí el cliente (navegador Web) hace una petición al servidor. El cliente es quien interpreta el HTML devuelto por el servidor, que únicamente lo transmite.

Ejemplo de este tipo de proceso:

* + En la primera consulta por parte del cliente hacia una web, se establece una conexión entre el servidor DNS y el ordenador que realiza la consulta o petición.

Este servidor DNS responde con la dirección IP del servidor web donde está alojado el contenido solicitado.

* + Después de solicita el contenido al servidor web mediante el protocolo HTTP/HTTPS.
  + Una vez que el servidor web ha recibido la solicitud del contenido solicitado por el cliente web, deberá procesar la solicitud hasta encontrar el contenido solicitado dentro del dominio correspondiente.
  + Envía el contenido solicitado al cliente web que lo solicitó.

Ejemplos de servidores web:

* + Servidor Microsoft IIS
  + Servidor HTTP Apache
  + Sun Java System web Server

**IP** =>

Una dirección IP, es un número único e irrepetible por el que se identifica a una computadora conectada a una red que utiliza el protocolo IP (Internet Protocol) en un modelo TCP/IP

Una dirección IP en un número binario que se suele expresar en forma decimal (IPv4) para facilitarnos su lectura o hexadecimal IPv6 que es otra version de IP

Ejemplo IPv4 => 8.8.8.8

Las direcciones IPv4, tienen 2 partes:

* + Los 2 primeros números indican la red
  + Los 2 últimos números el host concreto

Mascara de subred, indica que parte de la dirección es la de la red y que parte se refiere al host

**DNS** =>

Sistema de Nombres de Dominio

Sistema que se ocupa de la administración del espacio de nombres de dominio. Se encarga del proceso de traducción del nombre de dominio (que el usuario puede entender) en una dirección numérica (que la maquina puede entender)

Al indicar una url en el navegador web, este realiza una petición al llamado **resolver**, un componente especial del sistema operativo cuya función consiste en almacenar en caché direcciones IP ya solicitadas anteriormente, y proporcionarlas cuando la aplicación cliente (navegador, programa de correo) la solicita.

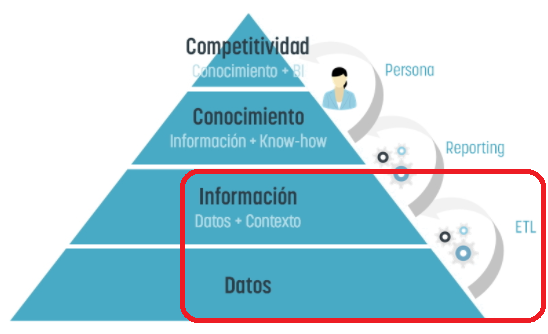
Si la dirección IP solicitada no se encuentra en el caché del resolver, este redirige la petición al servidor DNS que corresponda, que, en general, se trata del servidor DNS del proveedor de Internet

Servidor DSN, o servidor de nombres, es un software para servidores que consulta a la BBDD de un DNS, para traducir la dirección solicitada en una IP.

**TAREA 31**

**ETL** =>

ETL es el proceso que dará a los datos de los que se parte, un significado, un contexto, transformándolos en información.



https://blog.bi-geek.com/que-es-una-etl/

Las siglas que le dan nombre, identifican cada una de las fases de que se componen:

* + **E** => Extract.
    - Es el proceso de extracción de datos desde su sistema origen. En esta fase se hace una analisis de los datos extraídos, para comprobar si estos cumplen con el objetivo que se persigue y en caso de no ser así se descartan.
  + **T** => Transform
    - Este proceso realiza transformaciones sobre los datos extraídos, para darles formato y/o se realizan operaciones sobre ellos, de cara a que tengan tanto el contenido como el formato necesario para la analítica que se pretende aplicar sobre ellos (explotación de datos)
  + **L** => Load
    - Proceso de carga de los datos anteriores en el sistema destino, en el que se explotará la información, es decir, donde se realizará la analítica.

**Bases de Datos**  =>

Conjunto de datos pertenecientes a un mismo contexto, que son almacenados para su posterior uso.

Desde el punto de vista digital, se puede definir con un conjunto grande y estructurado de datos digitales organizados de manera tal que se permita la búsqueda, acceso y modificación de los datos que contiene.

Se pueden hacer distintos tipos de **clasificación** de las BBDD.

Si atendemos a la variabilidad de los datos que contienen:

* + **BBDD estáticas** =>

Son de solo lectura. Guardan datos históricos, que se pueden utilizar para analizar el comportamiento de sus datos en el tiempo. Utilizados en el mundo BI

* + **BBDD dinámicas** =>

Son de lectura y escritura. Permiten ademas de la consulta, operaciones de actualización, borrado y edición de los datos que contienen, por ejemplo, BBDD tras un ERP.

Si atendemos a la estructuración de su contenido:

* + **BBDD Relacionales** =>

Almacenan los datos relacionados en tablas. Cada table tiene una clave que le permite relacionarse de manera univoca con el resto de tablas en la BBDD.

Las tablas están compuestas por múltiples filas que contienen campos de información, en los cuales se insertan los datos.

Se debe definir previamente su modelo, es decir definir el esquema de funcionamiento que tendrán las tablas de forma anticipada.

Ejemplos: Oracle, MySQL, PostgreSQL

* + **BBDD NO Relacionales** =>

No tienen un identificador una clave que permita su relación con el resto de datos que contienen.

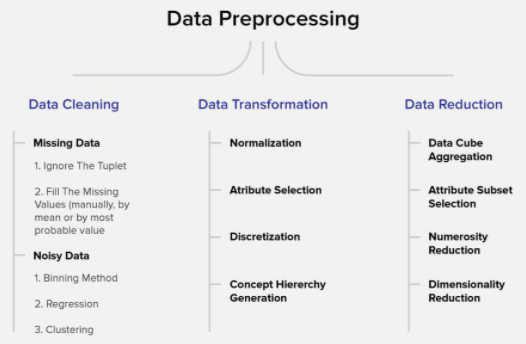
El formato de almacenamiento no es una tabla

Ejemplos: MongoDB, CouchDB, Cassandra, Neo4J

**Data Preprocessing** =>

Es el proceso que permite transformar los datos “crudos” del origen, en datos con un formato eficiente, con los que podamos trabajar.

Este preprocesado consta de distintos pasos:



**TAREA 33**

**Diagrama de flujo** =>

[Definición](#DiagramaDeFlujo) en tarea 21

**Bucles en programación** =>

[Definición](#Bucle) en tarea 22

**TAREA 34**

**API** => **A**pplication **P**rogramming **I**nterface

Conjunto de definiciones y protocolos, que permiten la comunicación entre aplicaciones, haciendo uso de las reglas definidas.

Las APIs permiten el uso de desarrollos de software, programas, que proveen de funcionalidades concretas (por ejemplo pago con PayPal o consulta de mapas de Google), que puede ser consultado/utilizado desde la aplicación que tu desarrolles, independientemente del lenguaje de programación utilizado en cualquier de los dos lados, evitando el tener que desarrollar nuevamente código para funcionalidades que ya estén disponibles.

**Protocolo de comunicación** =>

Conjunto de normas o reglas, que se deben seguir para que se pueda establecer la comunicación, entre diferentes elementos y que estos puedan intercambiar información.

**XML** => E**x**tensible **M**arkup **L**anguage

Formato para el envío de datos tradicionalmente usado. Se trata de un lenguaje basado en etiquetas, similar a HMTL



**JSON** => **J**ava**S**cript **O**bject **N**otation

Formato para el envío de datos más utilizado actualmente



XML **vs** JSON



JSON, más sencillo de entender (más claro) que XML

JSON, más ligero (menos bytes), menos peso de fichero con la misma información que XML

**TAREA 36**

**Base de Datos** =>

[Definición](#BaseDeDatos) en tarea 31

**SQL** => **S**tructured **Q**uery **L**anguage

Lenguaje de programación de consulta(read) y actualización(write) de BBDD relacionales.

SQL, se divide en 3 lenguajes:

* + ***DDL*** => **D**ata **D**efinition **L**anguage

Par definir las estructuras, tablas, registros, campos de los registros

* + ***DML*** => **D**ata **M**anipulation **L**anguage

Insert, Update, Delete, Select

* + ***DCL*** => **D**ata **C**ontrol **L**anguage

Con este se controla los permisos de acceso a las BBDDs

El lenguaje SQL, es estándar para cualquier BBDD relacional, aunque luego cada una tenga funciones propias.

**NoSQL** => **N**ot **O**nly **SQL**

BBDD NO Relacionales == NoSQL.

No tienen una estructura definida, no vamos a definir un modelo Entidad-relación. Podemos tener información duplicada.

En una BBDD relacional se normaliza su información (no hay datos duplicados) para mantener su regla de información no duplicada.

Las BBDDs NoSQL se crean para tener un acceso muy rápido a la información y ahorrar operaciones de cómputo (Amazon => DynamoDB o Facebook => Cassandra).

Podemos diferenciar las BBDD NoSQL por el tipo de datos que van a almacenar:

* + Clave – Valor
  + Documentales => MongoDB
  + Columnares => Cassandra / Hbase
  + Grafo => Neo4j
  + Series temporales => InfluxDB

**TAREA 38**

**Algoritmo** =>

[Definición](#Algoritmo) en tarea 11

**Diagrama de flujo** =>

[Definición](#DiagramaDeFlujo) en tarea 21

**Convención de programación** =>

Conjunto de normas o directrices a seguir para programar para dar calidad al software resultado. Estas directrices, se definen por cada lenguaje de programación y engloban distintos aspectos de la programación, comentarios, sangrías, buenas prácticas, convenciones de nombres, arquitectura, …

Es importante seguir estas convenciones, ya que seguramente el código que desarrollemos, en algún momento deberá ser mantenido por otra persona, por lo que cuanto más rápido y más en detalle lo entienda, menor resultará el coste de este mantenimiento.