

Materiales D.A promo C

Bienvenida

Si estás leyendo esto es porque has sido admitida en el Curso Intensivo de Adalab...

¡¡¡ Enhорабуена futura Analista de Datos !!!

Sabemos que estás deseando empezar a programar pero antes de meternos en materia te pedimos que hagas dos cosas.

Antes de empezar el curso debes:

- **Poner a punto tu ordenador instalando y configurando todos los programas** que utilizaremos durante el curso. Si tienes cualquier duda instalando los programas no te preocupes, el día de la bienvenida del curso dedicaremos dos horas a solucionar todos los problemas que hayas tenido. Para instalar los programas que utilizaremos durante el curso sigue las instrucciones que encontrarás en [Instalación de programas](#).
- La segunda tarea que te pedimos es que leas con detenimiento **nuestras guías** y veas lo vídeos que hay en ellas para adquirir los conocimientos básicos con los que empezaremos a trabajar durante el curso.
Te pedimos que prestes especial atención a las guías de [Agile](#) y [Scrum](#), ya que es importante que desde la primera semana del curso practiquemos estas dinámicas.

NOTA: Si encuentras erratas, faltas de ortografía, gramática, textos que se puedan explicar mejor o tienes cualquier sugerencia, por favor díñoslo a las profesoras. Estaremos encantadas de mejorar estos materiales.

Acerca del programa

Este repositorio recoge los materiales técnicos utilizados dentro del curso intensivo de Data Analytics impartido en Adalab. El curso está enfocado a mujeres sin conocimientos previos de programación.

La duración del programa es de 12 semanas. Se partirá del aprendizaje acerca de cómo están almacenados los datos, tipos de datos que existen y como extraerlos de sus fuentes, haciendo uso de SQL como lenguaje de programación para la manipulación y extracción de los datos. Por otro lado, en paralelo se aprenderá a hacer uso del lenguaje de programación Python, para la limpieza, procesado y exploración de los datos. Así como la creación de visualizaciones que permitan obtener conocimiento de los datos con los que se han trabajado, para dar soluciones a problemas de negocio.

Sobre Adalab

Adalab es una escuela especializada en formación digital para mujeres.

Trabajamos para formar y acompañar a mujeres que buscan un giro profesional adaptado a las nuevas necesidades digitales de las empresas.

A través de nuestro Curso Intensivo de Data Analytics las alumnas inician una carrera en tecnología, aportando diversidad al sector.

Objetivos generales

El objetivo del curso es que todas las participantes por un lado, sean autónomas en la extracción de información de bases de datos de forma eficiente, así como que tengan un conocimiento básico de la creación, modificación y mantenimiento de bases de datos.

Por otro lado, que las participantes se sientan seguras y cómodas, trabajando con Python para poder procesar, limpiar y extraer conocimiento.

Además contarán con conocimientos básicos para la realización de visualizaciones para acompañar al *story telling* de los proyectos en los que estén involucradas y ser capaces de usar el bagaje técnico para poder comunicarse de forma fluida con otras analistas de datos.

Por último, que puedan integrarse fácilmente en un equipo de trabajo bajo una filosofía ágil siguiendo marcos de trabajo como Scrum y haciendo uso de sistemas de control de versiones tales como Git.

Información y guías

Información del curso

Código de conducta en Adalab

Nuestro compromiso

En el interés de fomentar una comunidad abierta y acogedora, nosotros como contribuyentes y administradores nos comprometemos a hacer de la participación en nuestro proyecto y nuestra comunidad una experiencia libre de acoso para todos, independientemente de la edad, dimensión corporal, discapacidad, etnia, identidad y expresión de género, nivel de experiencia, nacionalidad, apariencia física, raza, religión, identidad u orientación sexual.

Nuestros estándares

Ejemplos de comportamiento que contribuyen a crear un ambiente positivo:

- Uso de lenguaje amable e inclusivo.
- Respeto a diferentes puntos de vista y experiencias.
- Aceptación de críticas constructivas.
- Enfocarse en lo que es mejor para la comunidad.
- Mostrar empatía a otros miembros de la comunidad.

Ejemplos de comportamiento inaceptable por participantes:

- Uso de lenguaje o imágenes sexuales y atención sexual no deseada.
- Comentarios insultantes o despectivos (*trolling*) y ataques personales o políticos.
- Acoso público o privado.
- Publicación de información privada de terceros sin su consentimiento, como direcciones físicas o electrónicas.
- Otros tipos de conducta que pudieran considerarse inapropiadas en un entorno profesional.

Nuestras responsabilidades

Los administradores del proyecto son responsables de clarificar los estándares de comportamiento aceptable y se espera que tomen medidas correctivas y apropiadas en respuesta a situaciones de conducta inaceptable.

Los administradores del proyecto tienen el derecho y la responsabilidad de eliminar, editar o rechazar comentarios, *commits*, código, ediciones de documentación, *issues*, y otras contribuciones que no estén alineadas con este Código de Conducta, o de prohibir temporal o permanentemente a cualquier colaborador cuyo comportamiento sea inapropiado, amenazante, ofensivo o perjudicial.

Alcance

Este código de conducta aplica tanto a espacios del proyecto como a espacios públicos donde un individuo esté en representación del proyecto o comunidad. Ejemplos de esto incluye el uso de la cuenta oficial de correo electrónico, publicaciones a través de las redes sociales oficiales, o presentaciones con personas designadas en eventos *online* u *offline*. La representación del proyecto puede ser clarificada explicitamente por los administradores del proyecto.

Aplicación

Ejemplos de abuso, acoso u otro tipo de comportamiento inaceptable puede ser reportado al equipo del proyecto en hola@adalab.es. Todas las quejas serán revisadas e investigadas, generando un resultado apropiado a las circunstancias. El equipo del proyecto está obligado a mantener confidencialidad de la persona que reportó el incidente. Detalles específicos acerca de las políticas de aplicación pueden ser publicadas por separado.

Administradores que no sigan o que no hagan cumplir este Código de Conducta pueden ser eliminados de forma temporal o permanente del equipo administrador.

Atribución

Este Código de Conducta es una adaptación del [Contributor Covenant](https://www.contributor-covenant.org/es/version/1/4/code-of-conduct.html), versión 1.4, disponible en <https://www.contributor-covenant.org/es/version/1/4/code-of-conduct.html>

Contenidos del curso

Metodología

La metodología de Adalab ha sido diseñada para que las alumnas adquieran las siguientes habilidades:

- Adquirir autonomía y destreza en la resolución de problemas para el desempeño del trabajo como frontend.
- Aprender a colaborar en un equipo siguiendo el marco de trabajo SCRUM y adquirir una mentalidad de mejora continua
- Saber plantear las dudas de forma clara.
- Resolver dudas de otras compañeras.
- Mejorar la gestión de tiempo y cumplimiento de horarios.

Datos básicos

- **Duración:** Curso intensivo de programación: 65 días de clase en total, aproximadamente 14 semanas, según días festivos y no lectivos.
- **Semana de la empleabilidad:** 5 días de desarrollo profesional y ayuda a las alumnas a posicionarse mejor a la hora de buscar trabajo.
- **Periodicidad:** clases diarias presenciales de lunes a viernes de 08.30 a 14.30 de programación.
- **Horas de formación:** Las horas lectivas del programa son 718 divididas en:
 - 423 horas de formación en programación tutorizada por Adalab
 - 260 horas de trabajo autónomo en programación por parte de la alumna con el material proporcionado por Adalab
 - 35 horas de sesiones y actividades para fomentar sus habilidades profesionales en filosofía Agile y Scrum, marca personal, herramientas de networking y empleabilidad
- **Equipo docente técnico:** Contamos con un equipo docente estable para la formación, reuniones de coordinación, evaluación de los ejercicios de las alumnas, creación y mejora de materiales, etc.
- **Facilitadores:** Para el resto de sesiones de desarrollo profesional contamos con vídeos y sesiones prácticas realizadas por los siguientes especialistas:
 - Especialista en filosofía agile y marco de trabajo scrum.
 - Especialista en búsqueda de empleo IT.
 - Especialista en Github.
 - Especialista en comunidades tecnológicas.
 - Especialista en mentoring.

Stack Tecnológico

Un stack tecnológico es el conjunto de tecnologías que juntas permiten desarrollar proyectos de análisis de datos, desde la obtención de la información, su procesado y hasta la creación de cuadros de mando

Una analista de datos debe aprender a manejar correctamente la sintaxis SQL y al menos un lenguaje de programación, así como la elaboración de gráficas adecuadas para poder transmitir a otras personas los resultados de los análisis.

En Adalab hemos elegido un stack basado en SQL y Python que incluye:

1. **MySQL:** Creación y gestión de bases de tablas, modificación, borrado y borrado lógico de registros. Extracción de datos.
2. **Python:** Para el tratamiento de datos.
 - Numpy: Tratamiento de datos numéricos y estadísticas
 - Pandas: Tratamiento de conjuntos de datos
 - Os: Tratamiento de ficheros
 - Matplotlib: Realización de gráficas
 - Seaborn: Realización de gráficas avanzadas
 - Sklearn: Resolver problemas de predicción y agrupamiento haciendo uso del aprendizaje automático.
3. **Git** para el control de versiones.
4. **Herramientas de trabajo y comunicación** de Slack, GitHub, GitHub projects, VSCode, Terminal.
5. **Agile y Scrum:** Experiencia en planificación y ejecución de proyectos bajo el marco de trabajo Scrum.

Clases del curso

El contenido de cada sesión por día se encuentra en estos materiales, las sesiones son las siguientes:

- Clases de contenidos
- Clases de pair programing
- Clases de proyecto
- Clases de resolución de dudas
- Clases de repaso
- Clases de desarrollo profesional.

Las sesiones técnicas habituales tienen la siguiente estructura y horario:

- **08.30 - 09.30:**
Kahoot + resumen de la sesión + resolución dudas de los ejercicios. Impartida por la profesora del bloque de contenido del módulo.
- **09.30 - 09.40:**
Descanso
- **09.40 - 10.40:**
Pair programming. Impartida por la profesora del bloque de contenido del módulo.
- **10.40 - 10.50:**
Descanso.
- **10.50 - 11.50:**
Kahoot + resumen de la sesión + resolución dudas de los ejercicios. Impartida por la profesora del bloque de contenido del módulo.
- **11.50 - 12.00:**
Descanso Pair programming. Impartidas por todas las profesoras.
- **12.00 - 13.00:**
Pair programming. Impartida por la profesora del bloque de contenido del módulo.
- **13.00 - 13.30:**
Descanso / Hora de la comida.
- **13.30 - 14.30:**
Proyecto. Impartida por la profesora del proyecto del módulo.
- **13.30 - 13.40:**
Descanso
- **13.40 - 15.30:**
Tutorías. Impartida por todas las profesoras de forma individual.
- **15:30 - ...:**
trabajo autónomo de las alumnas.

Clases de contenido

La clase de contenidos es la primera lección de cada día, se inicia a las 8.30 am en la sala principal de la promo. Durante todas las clases se aplica la clase invertida con el objetivo que vengas a esta clase con conocimientos previos.

Clase invertida

Clase invertida es una modalidad de aprendizaje mixto que pretende utilizar dos estrategias, la presencial y la virtual tomando en cada momento lo mejor de ellas. Por lo cual debes leer el temario en casa y realizar los ejercicios más sencillos y en clase se practican y desarrollan ejercicios con la ayuda de sus compañeras y la profesora.

Kahoot

Al comienzo de cada día tenemos un espacio para repasar los conocimientos de la sesión que habéis revisado anteriormente en casa de una forma divertida usando Kahoot. Pensar que el Kahoot es una forma de iniciar el día, no se evalúa y los profesores lo utilizamos para ver como orientar la clase y que temas han sido más difíciles de entender.

Intro a la sesión

Al comienzo de las sesiones se hace una breve intro de los contenidos a cubrir en la sesión, centrándose en la parte más compleja y se explican los conocimientos previos que deberías tener asimilados para ese día.

Realización de un resumen de la lección

Tras el kahoot se realiza un resumen colaborativo sobre los contenidos importantes de la lección, para facilitar el aprendizaje y recalcar los conocimientos clave de la misma. A la par que resolver las dudas que se tenga de los contenidos temáticos.

Resolución de los ejercicios que hayan generado dudas

Con el objetivo de que vean cómo se aplican los conocimientos de la lección se realizarán en vivo de forma colaborativa los ejercicios que hayan generado dudas con el fin de ver los puntos más complejos y resolver las dudas, con el fin de ver de forma práctica como se resolverían los ejercicios. Al final de cada semana se subirá a un repositorio la resolución, para que puedas consultarla en cualquier momento. Aunque no todas las lecciones tendrán una resolución realizada por las profesoras :)

Preguntas en clase

Para mantener el orden y la dinámica de la clase, se debe levantar la mano cuando tengamos una pregunta y una duda, y la profesora dará la palabra.

Clases de pair programing

Clases de pair programing

En pair programming dos personas se enfocan en desarrollar una serie o serie de ejercicios, para encontrar una solución más rápida y obtener mejores resultados. Cada programadora tiene un rol y existen diferentes técnicas y estilos para aplicarlo, y va a depender de diferentes factores como el contexto y el nivel de las programadoras. Se puede utilizar esta técnica en diferentes contextos, es una técnica de desarrollo ágil de software muy valorada en las empresas de desarrollo digital.

Pair programing en Adalab

Cada día en Adalab tienes dos sesiones de pair programing de una hora, posterior a la clase principal, donde realizarás el/los ejercicio/s propuesto/s para el módulo.

Durante las evaluaciones se revisarán los ejercicios de pair programming entregados con el fin daros feedback sobre el progreso

Es vital llevar al día los ejercicios de pair programming, ya que algunos de ellos estarán relacionados y será necesario haber completados los del día anterior

Durante esta sesión:

- Trabajas en mini salas de Zoom con la pareja que te han asignado las profesoras, tendrás una pareja nueva en cada sprint del curso.
- Puedes solicitar ayuda a los profesores a través de un hilo de Slack en el canal de programación correspondiente.

En las guías puedes acceder a la Guía de Pair Programming donde se explica en detalle las técnicas, beneficios, y cómo hacer pair programing.

Organización de las sesiones

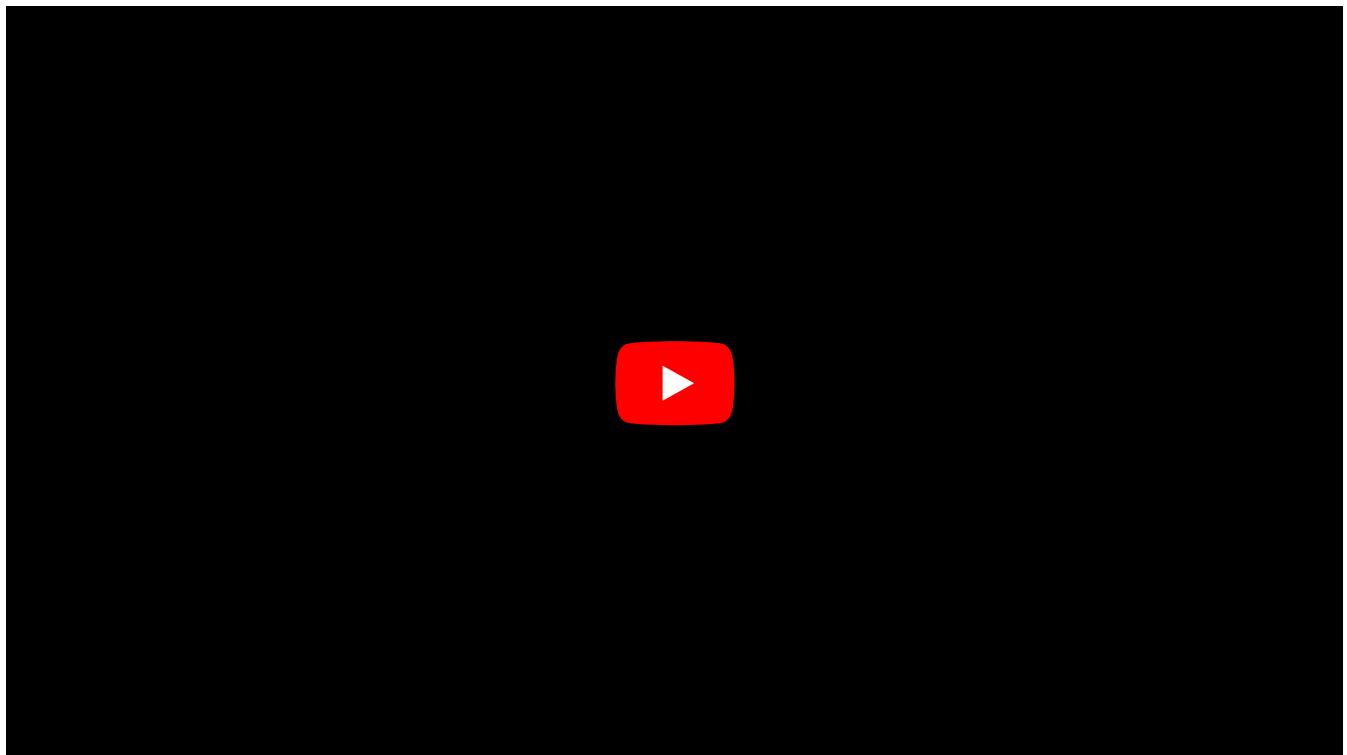
En cada sesión de pair programing os recomendamos:

1. Decidir con qué técnica o estilos de pair programing (**Driver-Navigator** o **Elastic**) se sienten más cómodas para utilizar con su pareja.
 - En caso de elegir **Driver-Navigator** alternar en cada ejercicio los roles. Las profes recomendamos la estrategia de **Driver-Navigator**.
2. Plantearse el objetivo del pair programing, en cada sesión resolveremos el o los diferentes ejercicios propuestos que buscan poner en práctica lo visto en clase, lo que se busca es lo siguiente:
 - Hacer lo correspondiente al ejercicio propuestos.
 - Repasar los ejercicios resueltos de la explicación en clase.
3. Decidir con qué herramienta y de que manera váis a trabajar en remoto en pareja, por ejemplo:
 - Utilizar la extensión Live Share y compartir el proyecto.
 - Compartir pantalla y solicitar el control de la pantalla de la compañera.
4. Preparar de antemano el dia previo el jupyter o archivo SQL necesario, con los enunciados incluidos, para poder ser más efectivas a la hora de comenzar la sesión de pair programming. De esto se encargará la que dicho dia tenga el rol de **Driver**. En el caso de utilizar la estrategia **elastic**, deberá encargarse una de las dos.

Herramientas

Una herramienta estupenda para programar en remoto es [Live Share de VS Code](#). Es uno de los plugins que os recomendamos instalar y que usaremos mucho durante el curso.

Aquí tenéis un [vídeo de cómo se usa](<https://www.youtube.com/watch?v=WFTGP6OIKD0>):



Clases de proyecto

En la actualidad, es muy necesario saber trabajar en equipo, además los proyectos y aplicaciones informáticas la mayoría de las veces son desarrolladas por un grupo de personas.

En Adalab hacemos como si fuera una empresa, trabajamos como en entorno de trabajo real, por lo que en los módulos de programación del curso vas a desarrollar un proyecto en equipo bajo el marco de trabajo Scrum (aprenderás que es Scrum en las próximas guías, no te preocupes ahora si no entiendes el término :)).

Organización

Cada día en Adalab tienes una sesión de proyecto de una hora y media, donde trabajarás en el proyecto que te corresponda con tu equipo.

Durante esta sesión:

- Trabajas en mini salas de Zoom con los miembros de tu equipo y tendrás un equipo diferente en cada módulo del curso.
- Puedes solicitar ayuda a los profesores a través de un hilo de Slack en el canal de programación correspondiente.

A continuación puedes ver un adelanto de los proyectos que harás durante el curso:

Proyecto 1. Obtención de ficheros desde las fuentes y tratamiento sencillo.

En este proyecto vamos a aprender a tratar una serie de archivos relacionados en diferentes formatos, concretamente texto, xml y de una base de datos. Para ello vamos a tener que ser capaz de abrir los ficheros, procesarlos y realizar algunas transformaciones sencillas sobre las variables de tipo texto. ¡Esta será vuestra primera experiencia de trabajo en equipo relacionada con programación! ¿Estáis preparadas?

Proyecto 2. Procesado de datos en crudo y realización de un informe.

En este proyecto vamos a aprender a tratar con los archivos que se han obtenido de el procesamiento de los ficheros del proyecto del módulo 1. Para ello tenemos que aprender a unificar diferentes fuentes de datos en un único fichero, aplicar la limpieza que nos parezca conveniente, transformar los datos y ser capaz de extraer conocimiento de la información contenida. Para ello vamos a tener que presentar un informe al final del módulo con los resultados obtenidos de forma visual y ser capaces de explicar algunas cosas interesantes del análisis realizado

Proyecto 3. Datathon de análisis de datos

Este proyecto sigue el estilo de una competición Kaggle o un Datathon (Un hackaton relacionado con el mundo data). Os daremos unos datos que tenéis que limpiar, hacer un EDA, interpretar lo que tenéis, y hacer unas regresiones para predecir una variable en base de una o más variables. En este caso es un poco especial porque vais a trabajar a competir de forma amistosa con el resto de equipos con el fin de optimizar una métrica específica de esta clase de modelos. ¡Sorpresa!

Clases de resolución de dudas y de repaso

Las clases de repaso son cuatro veces en cada módulo, siendo dos de ellas el día antes de la evaluación de los primeros contenidos y la evaluación del segundo bloque de contenidos. La dinámica de la sesión consiste en:

- Resolución de dudas pendientes del contenido de las clases o explicación extendida del temario más complejo.
- Realización de un ejercicio práctico de repaso, donde toda la clase va guiando al profesora y os sirve para prepararse para la evaluación.
- El objetivo es asegurarnos de que el contenido se ha entendido en su totalidad y que todas las alumnas tienen una base de conocimientos común.

Clases de desarrollo profesional

Adalab es un programa formativo-profesional donde el fin último es que conviertas en realidad tu principal objetivo: reinventarse profesionalmente y conseguir un puesto de trabajo como programadoras web.

Para ello, es necesario desarrollar habilidades técnicas y también habilidades sociales no técnicas, que equivalen a las actualmente llamadas Soft Skills.

El programa de desarrollo profesional en Adalab se enmarca dentro de la formación intensiva, con el objetivo de trabajar todas aquellas habilidades profesionales que son demandadas en el sector.

El entrenamiento y mejora de estas habilidades se lleva a cabo a través de actividades teórico / prácticas. Algunos de los objetivos de desarrollo profesional se trabajan en las sesiones técnicas de programación (ej: autonomía y resolución de problemas) y otras a través de vídeos o sesiones específicas de desarrollo profesional.

Dentro del curso intensivo tendrás algunas clases de desarrollo personal como la clase de Agile y Scrum y otras sesiones por la tarde. Posterior a finalizar el curso tendrás la semana de la empleabilidad.

Evaluaciones

Evaluación de bloque de contenido y entrevistas técnicas

Las evaluaciones de Adalab están pensadas para que el equipo docente pueda saber si las alumnas han adquirido los conocimientos mínimos de cada uno de los bloques del módulo, o deben volver a reevaluarse. Además, de cara a las alumnas están concebidas para que practiquen el formato de prueba de código + entrevista técnica que se encontrarán en muchos de los procesos de selección como programadoras.

En este caso se utilizará como base para la evaluación los ejercicios realizados por parejas durante las horas establecidas de pair programming, por lo tanto el dia previo a la evaluación se hará entrega de todos los ejercicios realizados para dicho bloque de contenido.

Las alumnas defenderán con su pareja los ejercicios entregados y se les podrá realizará una serie de preguntas técnicas sobre los conocimientos fundamentales para el avance adecuado de la formación del curso.

Para cada prueba establecemos unos criterios de evaluación que se dan a las alumnas junto con la prueba. Una vez entregada la profesora debe revisar cada prueba entregada para asignar una valoración para algunos de esos criterios. El resto de criterios se evalúan en la defensa de la prueba por pareja.

Las pruebas de evaluación de cada tendrán una duración de 40 minutos y se realizarán:

- Preguntas iniciales para asegurar la valoración de los criterios que se han revisado de forma previa (5 minutos).
- Se pide a la alumna algunas explicaciones sobre los ejercicios entregados o alguna serie de modificaciones sencillas sobre los diferentes ejercicios entregados para valorar el resto de criterios.
- Se da un feedback oral sobre los ejercicios (tanto del código entregado como entrevista técnica) y se comunica si se han superado los mínimos de aprendizaje del módulo (5 minutos).

Re-evaluación

Para las alumnas que no han superado los mínimos se realiza una re-evaluación durante el siguiente módulo. Se entrega un ejercicio nuevo para valorar con los mismos criterios que el de evaluación del bloque de contenido temático y que la alumna debe realizar de forma individual en tiempo fuera del horario de Adalab. La profesora realizará nuevamente la entrevista técnica. En caso de que alguna alumna no supere los objetivos mínimos de aprendizaje podrá continuar en el curso pero no podrá acceder a la Bolsa de Empleo de Adalab, puesto que a las empresas colaboradoras se les asegura unos conocimientos mínimos por parte de las alumnas.

Herramientas del curso

Avisos sobre las instalaciones

Si tienes problemas o errores durante el proceso de instalación no te preocunes, los primeros días las profes te ayudaremos a resolver estos problemas. Puedes intentar solucionarlo por tu cuenta, pero tampoco es necesario estar 20 intentando solucionar los posibles problemas que hayan surgido durante las instalaciones!

Instalación de ordenadores

Requerimientos mínimos del ordenador

En el curso de Data Analytics de Adalab necesitaremos usar un ordenador. Para poder realizar el curso de forma fluida, el ordenador debería tener estas características (no es obligatorio que las tenga, pero para que tengáis una orientación):

- Al menos 8GB de RAM (con 4GB irá algo más lento pero también se puede).
- Procesador i5 o similar con velocidad superior a 1GHz.
- Disco duro: recomendamos desde 128GB; si es SSD va a ir más fluido que si es HDD pero se puede tener más capacidad por menos precio.
- Tarjeta gráfica y pantalla: cualquiera pueden ir bien, aunque recomendamos pantalla de 15' o como mínimo 13' para poder programar junto a una compañera (con pantallas más pequeñas es posible pero se hace más complicado).

Para poder realizar el curso el sistema operativo (SO) del ordenador debe ser uno de los siguientes:

- **Windows 10**
- **Mac**
- **Ubuntu 18.04**

Windows 10 o Mac

Si utilizas Windows 10 o Mac ya tienes el sistema operativo y no necesitas hacer nada más. Puedes pasar al [siguiente paso](#).

Ubuntu 18.04 (linux)

NOTA: Si has solicitado un ordenador de Adalab, te lo daremos con Ubuntu instalado, así que puedes saltarte este apartado.

Si quieres trabajar en Ubuntu y no lo tienes instalado a continuación te explicamos cómo hacerlo. Hay dos opciones:

- Instalar Ubuntu borrando todo lo que hay en el ordenador (recomendada): esta opción es obligatoria para equipos viejos que no tienen recursos para tener a la vez Windows y Linux.
- Instalar Ubuntu junto a Windows: crearemos una partición del disco duro del ordenador para instalar Ubuntu y luego poder arrancar el ordenador desde el SO que elijamos. Esta opción es más compleja y depende del ordenador concreto que tengáis que pueda hacerse o no.

IMPORTANTE: Antes de proceder a la instalación es muy importante **hacer una copia de seguridad de los datos** que estén en el ordenador y que no queramos perder.

Usaremos la distribución de Ubuntu 18.04, que podemos descargar una ISO desde <http://releases.ubuntu.com/18.04/> y grabaremos en un CD o pendrive. El día de la sesión de bienvenida os ayudaremos a instalarlo. Seguiremos los pasos de instalación para instalar y configurar el sistema.

Si queremos mantener Windows, tendremos que hacer una partición:

- Hacer partición e instalar Ubuntu siguiendo este tutorial (mínimo de 30GB):
<https://www.tecmint.com/install-ubuntu-alongside-with-windows/>

NOTA: Al elegir instalar ubuntu, seleccionamos la opción de "opciones adicionales" para elegir en qué partición hacerlo. Una vez seleccionada la partición donde instalar Ubuntu, elegir que el gestor de arranque (bootloader) se instale en el disco duro principal en un desplegable abajo de la pantalla.

NOTA: En Adalab los profesores trabajamos con UBUNTU 18. Si quieras usar otra distribución de Linux, no podemos garantizar que sabremos resolver todos los problemas relacionados con el sistema operativo.

Posibles problemas en la instalación de Ubuntu

- En equipos de la marca MSI, [suele haber problemas con los drivers de teclado / ratón en la instalación, de la tarjeta gráfico y/o de la conexión WiFi.](#)
- En algunos Asus, no funciona la conexión Wifi, y [hay que instalar los drivers](#) que podéis pasar en un pendrive o con una conexión a Internet a través de vuestro móvil.
- También encontramos otras incompatibilidades de hardware (ordenador) con Ubuntu, [como esta que nos ha sucedido.](#)

Instalación de VS Code

¿Qué es VS Code?



VS Code

Visual Studio Code (también conocido como VS Code o Code a secas) es el editor de código con el que vamos desarrollar nuestras scripts y Jupyter notebooks haciendo uso del lenguaje de programación Python. Además dispone de un montón de extensiones y plugins que nos facilitan mucho la vida.

Instalación

macOS

Para Windows 10 y Mac [descarga e instala VS Code desde aquí](#).

Ubuntu 18

Desde Ubuntu, la forma más cómoda es acceder a la tienda Software de Ubuntu y ahí buscar **VS Code (llamado code en la tienda)** e instalarlo.

Windows 10

Para Windows 10 y Mac [descarga e instala VS Code desde aquí](#).

Plugins de VS Code

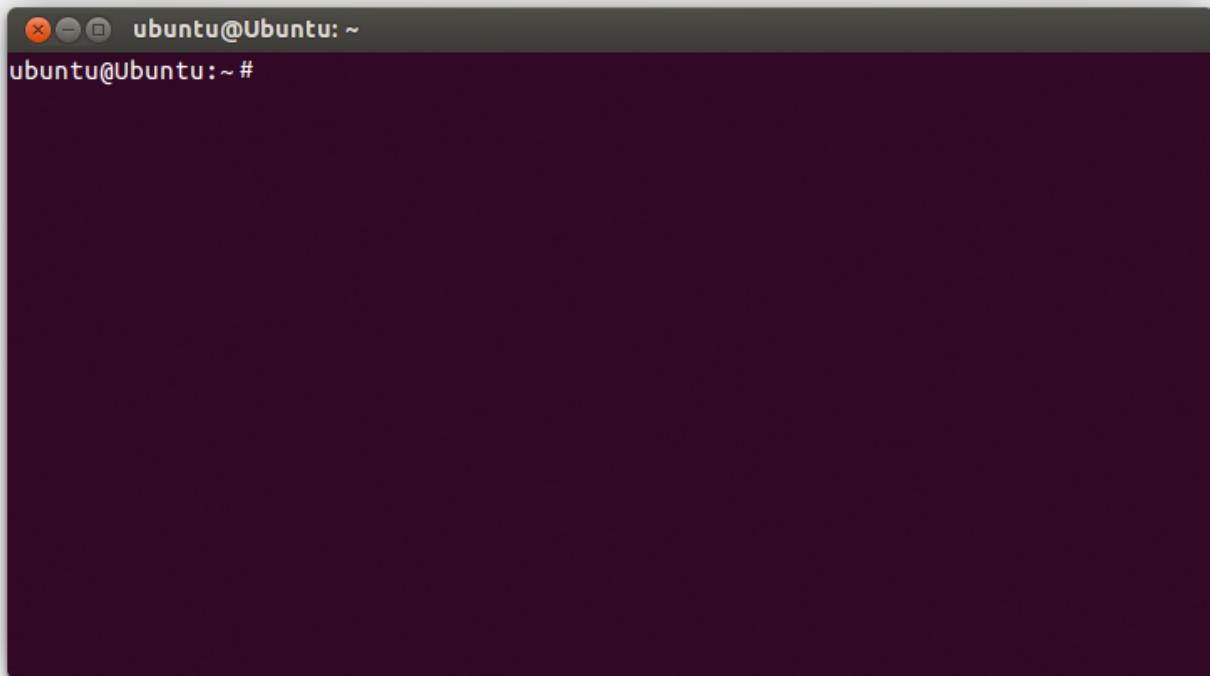
Los plugins son extensiones para ampliar y personalizar el funcionamiento de VS Code. Hay muchísimos. Nosotras vamos a utilizar las siguientes durante el curso. Accede a ellas e instálalas:

- [Python](#): para poder utilizar Python en VS Code. P.S: Es posible que las extensiones Jupyter y Pylance se instalen automáticamente al instalar esta extensión.
- [Jupyter](#): para poder utilizar los Jupyter notebooks dentro del entorno de VS Code.
- [Pylance](#): para poder utilizar los Jupyter notebooks dentro del entorno de VS Code.
- [Git Graph](#): para trabajar con Git.

Instalación de la terminal

Qué es la terminal

La terminal, también conocida como consola o shell, es una de nuestras herramientas principales en el desarrollo. Nos permite comunicarnos con el sistema operativo mediante pequeñas instrucciones de texto, **llamadas comandos**, sin necesidad de una interfaz gráfica. Tiene esta pinta:



Terminal de Ubuntu

Hay diferentes tipos de terminales, nosotras utilizaremos la terminal **Bash**.

Nota: antes de continuar debes conocer una peculiaridad de la terminal. Estamos acostumbrados a que cuando escribimos una contraseña en cualquier sitio, la contraseña aparece con asteriscos: *****. Sin embargo cuando la terminal nos pide que escribamos nuestra contraseña no pone asteriscos, en realidad no pone nada. Pero aún así sí estamos escribiendo la contraseña. Por ello cuando te pida la contraseña, escríbela y pulsa intro. Esto son frikadas de la programación.

Instalación de la terminal

macOS

Mac ya viene con una terminal bash instalada de serie. Por ello no necesitamos instalar nada más. Si quieres abrir tu terminal busca en el menú de Mac el programa **Terminal**.

Ubuntu 18

Ubuntu ya viene con una terminal bash instalada de serie. Por ello no necesitamos instalar nada más. Si quieres abrir tu terminal busca en el menú de Ubuntu el programa **Terminal**.

Windows 10

Windows tiene sus propias terminales como MS-DOS y Power Shell. Estas terminales **no** nos valen para programar así que vamos a instalar un mini Ubuntu dentro de Windows. Para ello, primero vamos a configurar Windows para que nos permita tener ese mini Ubuntu dentro de Windows:

1. Desde el menú inicio de tu Windows busca y abre **Activar o desactivar las características de Windows**. (En inglés: Turn Windows features on or off.)
2. Activa la opción **Subsistema de Windows para Linux**. Acepta y reinicia.

Una vez terminado debes instalar Ubuntu desde la Microsoft Store:

1. Desde el menú inicio de tu Windows busca y abre **Microsoft Store**.
2. En el buscador del store busca **Ubuntu**.
3. Instala **Ubuntu 18.04 LTS**.
4. Verás que en tu menú inicio se habrá añadido un programa llamado **Ubuntu 18.04 LTS**, ábrelo (la primera vez puede tardar 1 ó 2 minutos).
5. Te pedirá que añadas un nuevo usuario y una nueva contraseña (Enter new UNIX username, New password y Retype new password). **Esta contraseña es la que utilizaremos para instalar programas en la terminal.** ¡No la pierdas! Cada vez que tengas que instalar algo a través de la terminal tendrás que escribir esta contraseña.

Después de haber hecho estas dos cosas tu Windows contará con una **terminal** que funciona igual que si estuvieras trabajando en Linux / Ubuntu.

A continuación vamos a configurar **VS Code** para que siempre trabaje con esta terminal:

1. Abre la configuración de VS Code pulsando en el icono de la tuerca (esquina inferior izquierda)  y a continuación **Settings**.
2. Busca la opción **Terminal > External: Windows exec** (esríbelo en la caja de búsqueda de la parte superior o entra en el menú **Features** y luego **Terminal**).
3. Reemplaza el texto por **C:\Windows\System32\wsl.exe**.

4. Abre una terminal pulsando en el menú superior > **Terminal** > **New terminal**: una nueva terminal se abrirá en la parte inferior de VS Code.
5. En dicha parte inferior hay un desplegable, ábrelo y pulsa en **Select default profile**.
6. Selecciona la opción **Ubuntu (WSL)**.

Problemas al instalar la terminal en Windows 10

A veces desde Windows 10 abrimos una terminal en VS Code y esta aparece vacía, no aparece ningún texto.

Nos hemos dado cuenta que algunos antivirus bloquean la terminal. **Nuestra recomendación es desinstalar el antivirus y reiniciar el ordenador**. En el 99% de los casos se soluciona.

Si aún así, la terminal sigue sin funcionar, **te ayudaremos a que funcione** durante los primeros días de clase.

Usar la terminal dentro de VS Code

En Mac, Windows y Ubuntu tenemos dos formas de utilizar la terminal:

- Buscándola en el menú del sistema operativo y abriéndola o
- Desde dentro de VS Code.

Preferimos hacerlo desde dentro de VS Code porque es más cómodo. Por ejemplo las opciones de copiar y pegar funcionan mejor, la terminal siempre se abre en la misma carpeta en la que hemos abierto el VS Code...

Por ello cada vez que tengamos que usar la terminal tendremos que:

1. Abrir VS Code en la carpeta en la que queramos trabajar.
2. Pulsar en el **menú superior > Terminal**.
3. Pulsar en **Nueva terminal**.

Instalar programas a través de la terminal

De vez en cuando vamos a instalar programas a través la terminal. Para ello tendremos que escribir comandos en la terminal como:

```
apt install mysql-server
```

Es probable que al instalar programas la consola muestre un error diciendo que no tenemos permisos para instalarlo porque se requiere ser administrador para poder hacerlo. En los errores suele poner algo como EACCES (error de acceso) o (13: Permission denied) :

```
adalab@DESKTOP-GTGTDPK:/mnt/c/Users/alex$  
(base) adalab@DESKTOP-GTGTDPK:/mnt/c/Users/alex$ apt install mysql-server  
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)  
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?  
(base) adalab@DESKTOP-GTGTDPK:/mnt/c/Users/alex$
```

Error en la terminal

Cuando nos pase esto y sepamos que el error es porque se necesitan permisos de administración para instalar el programa, **usaremos el comando mágico `sudo`**.

Instalar con `sudo`

El comando `sudo` nos permite instalar programas con permisos de administrador. Para ello escribiremos el comando `sudo` delante del comando que queramos ejecutar, por ejemplo:

```
sudo apt install mysql-server
```

A continuación la terminal nos pedirá la contraseña del administrador.

La contraseña en Mac y Ubuntu **es la de nuestro usuario**, la que escribimos al iniciar sesión cuando arrancamos el ordenador. La contraseña en Windows **es la que hemos escrito cuando hemos instalado el mini Ubuntu** al principio de esta página.

IMPORTANTE: acuérdate del comando mágico `sudo`, lo utilizarás a menudo en tu vida como programadora.

Instalación de Anaconda

Qué es Anaconda?



Anaconda

Anaconda es una distribución libre y abierta de los lenguajes de programación Python y R. En nuestro caso se hará uso únicamente de Python. Empleamos esta distribución ya que incluye una serie de paquetes muy útiles y de uso común.

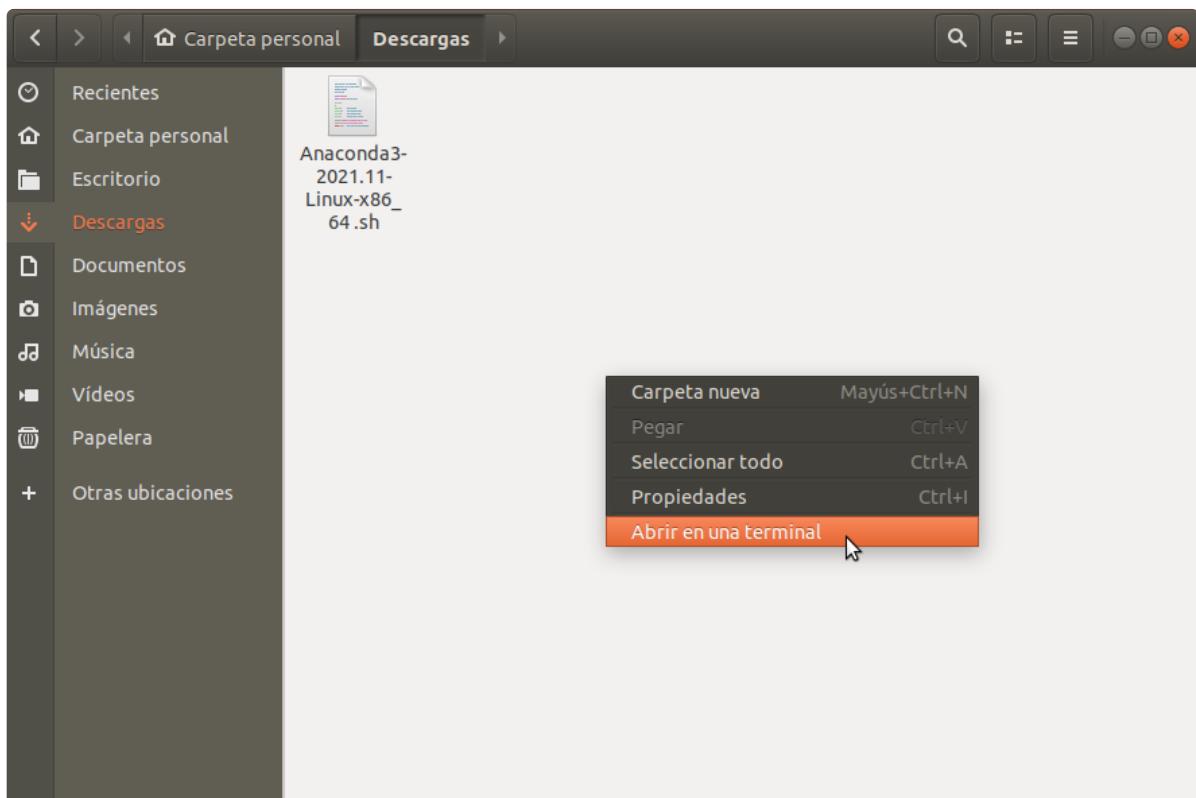
Instalación

macOS

1. Descarga el [instalador Anaconda Mac](#).
2. Abre el archivo descargado y instala la aplicación como harías normalmente (Total, nadie se lee los términos y condiciones ¿Verdad?)

Ubuntu 18

1. Descarga el [instalador Anaconda Ubuntu](#).
2. Accede a la carpeta de descargas, haz click secundario del ratón y selecciona la opción Abrir en una terminal.



3. Escriba en la terminal:

```
bash Anaconda3-2021.11-Linux-x86_64.sh
```

```
alejandro@alejandro-GL62M-7REX: ~/Descargas
Archivo Editar Ver Buscar Terminal Ayuda
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

alejandro@alejandro-GL62M-7REX:~/Descargas$ bash Anaconda3-2021.11-L
inux-x86_64\ .sh
```

A screenshot of a terminal window in Ubuntu 18.04. The terminal prompt shows the user is in the 'Descargas' directory. The command 'bash Anaconda3-2021.11-Linux-x86_64.sh' is entered and ready to be executed. The terminal interface includes a dark background with light-colored text and standard Linux-style icons.

4. Presiona la tecla de enter hasta que te pida que escribas 'yes' or 'no'. Escribiendo 'yes'. Tras esto una vez termine la instalación, preguntará si deseas iniciar Anaconda by running conda init?. Escribe 'yes'.

```
alejandro@alejandro-GL62M-7REX: ~/Descargas
Archivo Editar Ver Buscar Terminal Ayuda
Please answer 'yes' or 'no':'
>>>
Please answer 'yes' or 'no':'
>>> yes

Anaconda3 will now be installed into this location:
/home/alejandro/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/alejandro/anaconda3] >>>
PREFIX=/home/alejandro/anaconda3
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
```

Windows 10

1. Abre VS code y una terminal.
2. Copia la siguiente linea y ejecútala en la terminal:

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh
```

3. Una vez descargado, copia la siguiente línea y ejecútala:

```
bash Anaconda3-2021.11-Linux-x86_64.sh
```

4. Presiona la tecla de enter hasta que te pida que escribas 'yes' or 'no'. Escribiendo 'yes'. Tras esto una vez termine la instalación, preguntará si deseas iniciar Anaconda by running conda init?. Escribe 'yes'.

```
alejandro@alejandro-GL62M-7REX: ~/Descargas
```

```
Archivo Editar Ver Buscar Terminal Ayuda
Please answer 'yes' or 'no':'
>>>
Please answer 'yes' or 'no':'
>>> yes

Anaconda3 will now be installed into this location:
/home/alejandroadalab/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/alejandroadalab/anaconda3] >>>
PREFIX=/home/alejandroadalab/anaconda3
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
```

Enhorabuena, has instalado Anaconda!!

Instalación de Git

Qué es Git



Git

Git es un sistema de **control de versiones** de código además de una **herramienta para compartir** código.

El control de versiones nos sirve para ver el histórico de cambios que hemos hecho en el código. Es útil pasar saber por ejemplo que hace un mes nuestra compañera Mari Carmen añadió un determinado código al proyecto. **Git nos dice quién, cuándo, por qué y dónde se cambió cada línea de código de un proyecto.**

También es una herramienta para compartir código entre nuestras compañeras de programación de los proyectos de Adalab, nuestros profes o la empresa en la que trabajaremos. **Es como el Drive o Dropbox de los programadores.**

Qué es GitHub

Si Git es el sistema o programa para compartir código, [GitHub](#) es una de tantas empresas que hay para usar Git. A través de su página podemos crear proyectos y compartirlos con otras personas. También funciona como red social de programación.

¡¡¡Gracias a Git y GitHub vamos a poder trabajar fácilmente en remoto desde el primer día!!!

Instalación de Git

Lo primero que tenemos que hacer es instalar Git en nuestro ordenador.

macOS

Descarga e instala [Git para Mac](#).

Ubuntu 18

Abre una terminal y ejecuta las siguientes líneas de una en una:

```
sudo add-apt-repository ppa:git-core/ppa
```

```
sudo apt update
```

```
sudo apt install git
```

Windows 10

Abre una terminal en VS Code y ejecuta las siguientes líneas de una en una:

```
sudo add-apt-repository ppa:git-core/ppa
```

```
sudo apt update
```

```
sudo apt install git
```

Por último descarga e instala [Git para Windows](#).

Descarga e instala [Git para Mac](#).

Comprobar si lo hemos instalado bien

Una vez terminada la instalación de Git desde cualquiera de los 3 sistemas operativos debemos comprobar que todo ha ido bien. Para ello ejecutaremos en la terminal la siguiente línea:

```
git --version
```

Y la terminal debe mostrar la versión de Git instalada, algo como `git version 2.17.1`. Si por el contrario la terminal muestra el mensaje `No se ha encontrado la orden «git»...` es que algo hemos hecho mal. Vuelve a repetir todos los pasos y si no te funciona consulta a tu profesor.

Configuración de Git

Una vez instalado Git tenemos que crearnos una cuenta en GitHub y configurar nuestro ordenador para que **el Git de nuestro ordenador y nuestra cuenta de GitHub estén enlazados**:

1. Entra en [GitHub](#).
2. Crea una cuenta **con el email que has facilitado a Adalab**.
3. Cuando elijas tu nombre de usuaria ten en cuenta que las empresas te pedirán tu perfil de GitHub para ver cómo programas. No pongas un nombre de usuario demasiado informal.
4. Guarda la contraseña que elijas ya que la usaremos (bastante) más adelante.
5. Configura Git en tu ordenador:

Configuración del nombre y email

En los 3 sistemas operativos tenemos que configurar el nombre de usuario y el email. Para ello abre una terminal y escribe los siguientes comandos uno por uno. Sustituye el nombre y el email por tus datos:

```
git config --global user.name "María del Carmen"
```

```
git config --global user.email "lamary@gmail.com"
```

Configuración

macOS

Para poder almacenar la contraseña de GitHub en Mac, abrimos una terminal y escribimos el siguiente comando:

```
git config --global credential.helper osxkeychain
```

Una vez hayamos realizado ese paso, no necesitaremos hacer ningún cambio más.

Ubuntu 18

Para poder almacenar la contraseña de GitHub en Ubuntu, abrimos una terminal y escribimos los siguientes comandos uno por uno:

```
sudo apt-get install libsecret-1-0 libsecret-1-dev
```

```
cd /usr/share/doc/git/contrib/credential/libsecret
```

```
sudo make
```

```
cd -
```

```
git config --global credential.helper /usr/share/doc/git/contrib/credential/libsecr
```

Al hacer esto, la próxima vez que introduzcamos nuestra contraseña de GitHub, esta se almacenará de forma segura en nuestro ordenador y no será necesario volver a introducirla de nuevo.

Windows 10

Para poder almacenar la contraseña de GitHub en Windows, abrimos una terminal y escribimos los siguientes comandos uno por uno:

```
sudo apt-get install libsecret-1-0 libsecret-1-dev
```

```
cd /usr/share/doc/git/contrib/credential/libsecret
```

```
sudo make
```

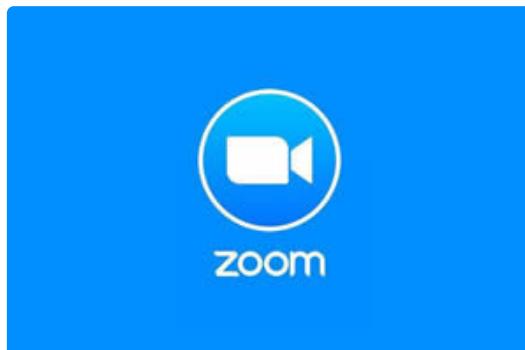
```
cd -
```

```
git config --global credential.helper "/mnt/c/Program\ Files/Git/mingw64/libexec/gi
```

Al hacer esto, la próxima vez que introduzcamos nuestra contraseña de GitHub en VS Code, esta se almacenará de forma segura en nuestro ordenador y no será necesario volver a

introducirla de nuevo.

Instalación de Zoom



Zoom

En Adalab utilizamos varias herramientas para comunicarnos. Una de ellas es la plataforma de video streaming Zoom, así que vamos instalarlo en nuestro ordenador y crearnos una cuenta.

Instalación

A continuación debes instalar Zoom en tu ordenador:

macOS

- Descarga e instala [Zoom](#).

Ubuntu 18

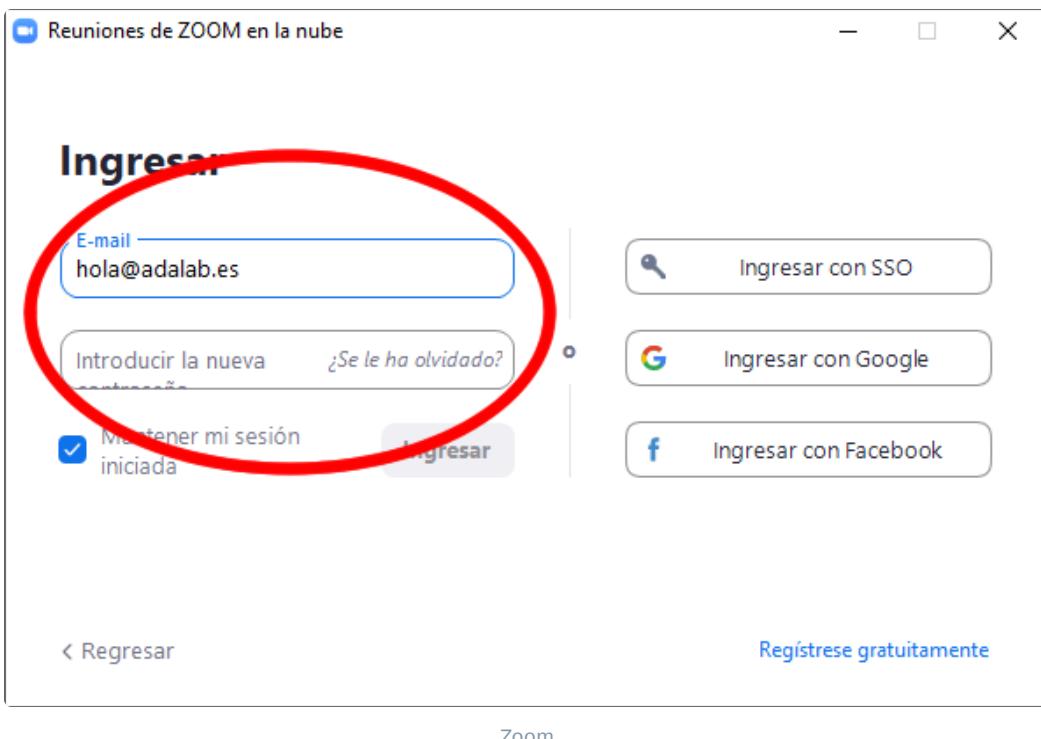
- Accede a la tienda Software de Ubuntu.
- Buscar **Zoom** e instálalo.

Windows 10

- Descarga e instala [Zoom](#).

Crea una cuenta en Zoom

- Entra en [Zoom.us](#).
- Crea una cuenta **con el mismo email que nos has proporcionado** a Adalab.
- Abre la aplicación de Zoom de tu ordenador.
- Ingresa desde el formulario de la izquierda. Esto es importante, si accedes desde el botón de la derecha **Ingresar con Google** hay cosas que no funcionan correctamente.



Consejos

- Descarga la aplicación. La versión web tiene menos funcionalidades que la aplicación.
- Usa auriculares, al ser posible con micrófono, para reducir el eco.
- Usa la webcam. Al vernos las caras la comunicación fluye mucho mejor.

Instalación de Slack



Slack

Otra herramienta muy buena para trabajar en remoto es **Slack**. En Adalab la usamos para todo. Un par de días antes de comenzar el curso te enviaremos un enlace para que accedas al Slack de Adalab.

Puedes usar su versión web o instalar su aplicación. Te pedimos que instales la aplicación ya que en la versión web hay cosas que no se pueden hacer.

Para descargarlo desde Windows y Mac [entra aquí](#).

Desde Ubuntu, la forma más cómoda es acceder a la tienda Software de Ubuntu y ahí buscar **Slack** e instalarlo.

Instalación de MySQL

Qué es MySQL?



MySQL es un sistema de gestión de bases de datos relacionales (en inglés: Relational Data Base Management System o RDBMS) de código abierto respaldado por Oracle y basado en el lenguaje de consulta estructurado (en inglés: Structured Query Language o SQL). Lo utilizaremos para almacenar y trabajar con bases de datos relacionales, que a efectos prácticos no es más que conjuntos de tablas que contienen información y que están relacionadas entre sí.

Instalación

macOS

1. Descarga [MySQL Server](#).
2. Abre el archivo descargado y instala la aplicación como harías normalmente.
3. Durante el proceso de instalación te pedirá la contraseña para la usuaria **root**, elige como contraseña **AlumnaAdalab**.
4. Descarga [MySQL Workbench](#).
5. Abre el archivo descargado y arrastra el ícono a la carpeta de aplicaciones.
6. Busca y abre ahora MySQL Workbench, automáticamente te debería haber detectado un servidor con nombre **LocalInstance 3306**, ve al cuadro de dicha aplicación y dale click al botón derecho del ratón. Selecciona **Edit connection** y en **Connection name** escribe **Adalab Server** y dale al botón Close.
7. Ahora haz click sobre el recuadro donde pone Adalab Server, introduce la contraseña de la usuaria **root** (recordamos que es **AlumnaAdalab**). De esta manera el programa nos llevará a la pestaña principal de nuestro MySQL Server.

Ubuntu 18

1. Abre una terminal y escribe:

```
sudo apt-get update
```

Tras esto:

```
sudo apt-get upgrade
```

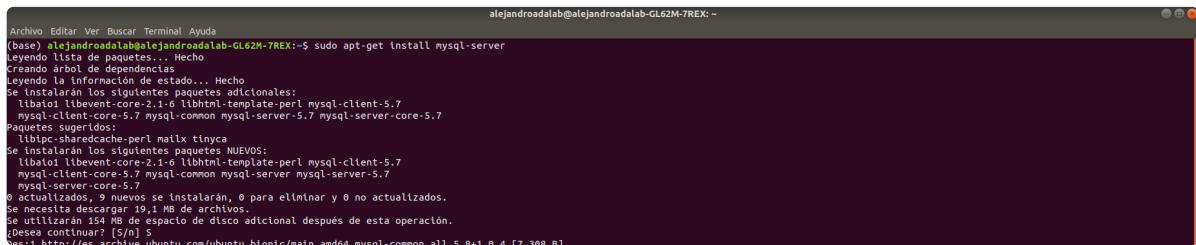
Finalmente:

```
sudo apt-get autoremove
```

De esta forma nos aseguraremos de instalar la última versión disponible de MySQL y MySQL Workbench.

2. Tras el proceso anterior, escribimos en la consola:

```
sudo apt-get install mysql-server
```



```
alejandro@alejandro-GL62M-7REX: ~
(base) alejandro@alejandro-GL62M-7REX: ~$ sudo apt-get install mysql-server
Leyendo lista de paquetes... Hecho
Creando índice de paquetes... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libatomic1 libevent-core-2.1-6 libhtml-template-perl mysql-client-5.7
  mysql-client-core-5.7 mysql-common mysql-server-5.7 mysql-server-core-5.7
Paquetes sugeridos:
  libhtml-template-perl mailx tinc
Se instalarán los siguientes paquetes NUEVOS:
  libatomic1 libevent-core-2.1-6 libhtml-template-perl mysql-client-5.7
  mysql-client-core-5.7 mysql-common mysql-server mysql-server-5.7
  mysql-server-core-5.7
0 actualizados, 9 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se instalarán 154 MB de archivos.
Se utilizarán 154 MB de espacio de disco adicional después de esta operación.
(Desea continuar? [S/n]) S
Descargando http://es.archive.ubuntu.com/ubuntu/bionic/main amd64 mysql-common all 5.8+1.0.4-7.308.B1
```

3. Llegados a este punto, el programa nos preguntará si deseamos continuar con la instalación, decimos que sí.

alejandro@alejandro-GL62M-7REX: ~

```

Archivo Editar Ver Buscar Terminal Ayuda
(base) alejandroadalab@alejandroadalab-GL62M-7REX:~$ sudo apt-get install mysql-server
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libaio1 libevent-core-2.1-6 libhtml-template-perl mysql-client-5.7
  mysql-client-core-5.7 mysql-common mysql-server-5.7 mysql-server-core-5.7
Paquetes sugeridos:
  libipc-sharedcache-perl mailx tinyca
Se instalarán los siguientes paquetes NUEVOS:
  libaio1 libevent-core-2.1-6 libhtml-template-perl mysql-client-5.7
  mysql-client-core-5.7 mysql-common mysql-server mysql-server-5.7
  mysql-server-core-5.7
0 actualizados, 9 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 19,1 MB de archivos.
Se utilizarán 154 MB de espacio de disco adicional después de esta operación.
Desea continuar? [S/n] S
Des:1 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 mysql-common all 5.8+1.0.4 [7.308 B]
Des:2 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libaio1 amd64 0.3.110-Subuntu0.1 [6.476 B]
Des:3 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 mysql-client-core-5.7 amd64 5.7.36-0ubuntu0.18.04.1 [6.6
38 kB]
Des:4 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 mysql-client-5.7 amd64 5.7.36-0ubuntu0.18.04.1 [1.942 kB
]
Des:5 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 mysql-server-core-5.7 amd64 5.7.36-0ubuntu0.18.04.1 [7.4
26 kB]
Des:6 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 libevent-core-2.1-6 amd64 2.1.8-stable-4build1 [85,9 kB]
Des:7 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 mysql-server-5.7 amd64 5.7.36-0ubuntu0.18.04.1 [2.906 kB
]
Des:8 http://es.archive.ubuntu.com/ubuntu bionic/main amd64 libhtml-template-perl all 2.97-1 [59,0 kB]
Des:9 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 mysql-server all 5.7.36-0ubuntu0.18.04.1 [9.944 B]
Descargados 19,1 MB en 2s (9.349 kB/s)
Preconfigurando paquetes ...
Seleccionando el paquete mysql-common previamente no seleccionado.
(Leyendo la base de datos ... 175208 ficheros o directorios instalados actualmente.)

```

4. Una vez termine la instalación, escribimos lo siguiente para comprobar si se ha instalado correctamente:

```
sudo mysql
```

```

Archivo Editar Ver Buscar Terminal Ayuda
Desempaquetando libhtml-template-perl (2.97-1) ...
Seleccionando el paquete mysql-server previamente no seleccionado.
Preparando para desempaquetar .../mysql-server_5.7.36-0ubuntu0.18.04.1_all.deb ...
Desempaquetando mysql-server (5.7.36-0ubuntu0.18.04.1) ...
Configurando libevent-core-2.1-6:amd64 (2.1.8-stable-4build1) ...
Configurando libhtml-template-perl (2.97-1) ...
Configurando libaio1:amd64 (0.3.110-Subuntu0.1) ...
Configurando mysql-client-core-5.7 (5.7.36-0ubuntu0.18.04.1) ...
Configurando mysql-server-core-5.7 (5.7.36-0ubuntu0.18.04.1) ...
Configurando mysql-client-5.7 (5.7.36-0ubuntu0.18.04.1) ...
Configurando mysql-server-5.7 (5.7.36-0ubuntu0.18.04.1) ...
update-alternatives: utilizando /etc/mysql/mysql.cnf para proveer /etc/mysql/my.cnf (my.cnf) en modo automático
Renaming removed key_buffer and myisam-recover options (if present)
Created symlink /etc/systemd/system/multi-user.target.wants/mysql.service → /lib/systemd/system/mysql.service.
Configurando mysql-server (5.7.36-0ubuntu0.18.04.1) ...
Procesando disparadores para libc-bin (2.27-3ubuntu1.4) ...
Procesando disparadores para systemd (237-3ubuntu10.52) ...
Procesando disparadores para man-db (2.8.3-2ubuntu0.1) ...
Procesando disparadores para ureadahead (0.100.0-21) ...
ureadahead will be reproffed on next reboot
(base) alejandroadalab@alejandroadalab-GL62M-7REX:~$ sudo mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.36-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

5. Vamos ahora a realizar la configuración del usuario root de MySQL. En la misma terminal que teníamos abierta escribimos:

```
sudo mysql -u root -p
```

```

(base) alejandroadalab@alejandroadalab-GL62M-7REX:~$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.36-0ubuntu0.18.04.1 (Ubuntu)
Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Veremos que ahora la terminal ha cambiado, apareciéndonos en cada línea nueva al escribir:

```
mysql>
```

Tras esto escribimos:

```
use mysql
```

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
```

6. A continuación ponemos:

```
SELECT User, Host, plugin FROM mysql.user;
```

Veremos que nos ha aparecido una tabla con la lista de usuarios y algunos parámetros de su configuración.

```
mysql> SELECT User, Host, plugin FROM mysql.user;
+-----+-----+-----+
| User | Host | plugin |
+-----+-----+-----+
| root | localhost | auth_socket |
| mysql.session | localhost | mysql_native_password |
| mysql.sys | localhost | mysql_native_password |
| debian-sys-maint | localhost | mysql_native_password |
+-----+-----+-----+
4 rows in set (0,00 sec)
```

7.Tras lo anterior escribimos:

```
UPDATE user SET plugin='mysql_native_password' WHERE User='root';
```

Y vemos que ha cambiado una fila de la tabla.

```
mysql> UPDATE user SET plugin='mysql_native_password' WHERE User='root';
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

8.Escribimos ahora:

```
FLUSH PRIVILEGES;
```

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,00 sec)
```

9.Comprobamos ahora que se ha realizado el proceso correctamente, escribiendo:

```
SELECT User, Host, plugin FROM mysql.user;
```

Y nos debería de quedar lo siguiente:

```
mysql> SELECT User, Host, plugin FROM mysql.user;
+-----+-----+-----+
| User | Host | plugin |
+-----+-----+-----+
| root | localhost | mysql_native_password |
| mysql.session | localhost | mysql_native_password |
| mysql.sys | localhost | mysql_native_password |
| debian-sys-maint | localhost | mysql_native_password |
+-----+-----+-----+
4 rows in set (0,00 sec)
```

10. Como penúltimo paso, vamos a establecer la contraseña para la usuaria `root` (ya que la primera vez, no nos la ha pedido), que será necesaria para poder acceder posteriormente a la base de datos. Escribimos lo siguiente para salir de la base de datos:

```
exit;
```

Ahora habremos regresado a la terminal original y escribimos lo siguiente:

```
mysqladmin -u root password AlumnaAdalab
```

Tras esto ya podremos utilizar MySQL con la usuaria `root` y contraseña `AlumnaAdalab`.

```
mysql> SELECT User, Host, plugin FROM mysql.user;
+-----+-----+-----+
| User | Host | plugin |
+-----+-----+-----+
| root | localhost | mysql_native_password |
| mysql.session | localhost | mysql_native_password |
| mysql.sys | localhost | mysql_native_password |
| debian-sys-maint | localhost | mysql_native_password |
+-----+-----+-----+
4 rows in set (0,00 sec)
```

11. Llegados a este punto, cerramos la terminal y abrimos una nueva para proceder con la instalación de MySQL Workbench. En la nueva terminal escribimos:

```
sudo snap install mysql-workbench-community
```

Si apareciera un error indicando que no puede encontrar el comando snap, introduce lo siguiente y vuelve a intentarlo una vez se haya instalado snap

```
sudo apt install snap
```

```
Archivo Editar Ver Buscar Terminal Ayuda
(base) alejandroadalab@alejandroadalab-GL62M-7REX:~$ sudo snap install mysql-workbench-community
[sudo] contraseña para alejandroadalab:
Se ha instalado mysql-workbench-community 8.0.25 por Tonin Bolzan (tonybolzan)
(base) alejandroadalab@alejandroadalab-GL62M-7REX:~$
```

12. Finalmente, abrimos una terminal nueva y escribimos lo siguiente para asegurarnos que MySQL Workbench pueda conectarse de forma adecuada con la base de datos de MySQL:

```
sudo snap connect mysql-workbench-community:password-manager-service
```

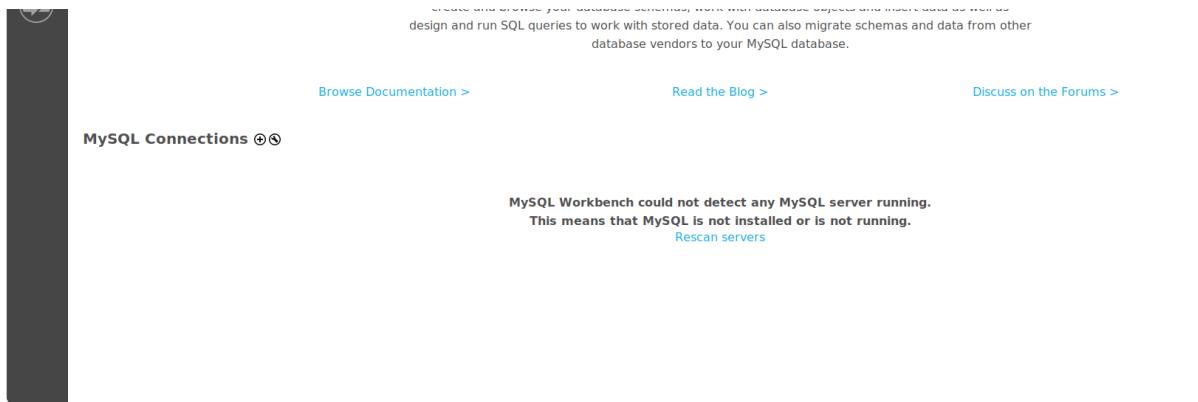
Y tras esto:

```
sudo snap connect mysql-workbench-community:ssh-keys
```

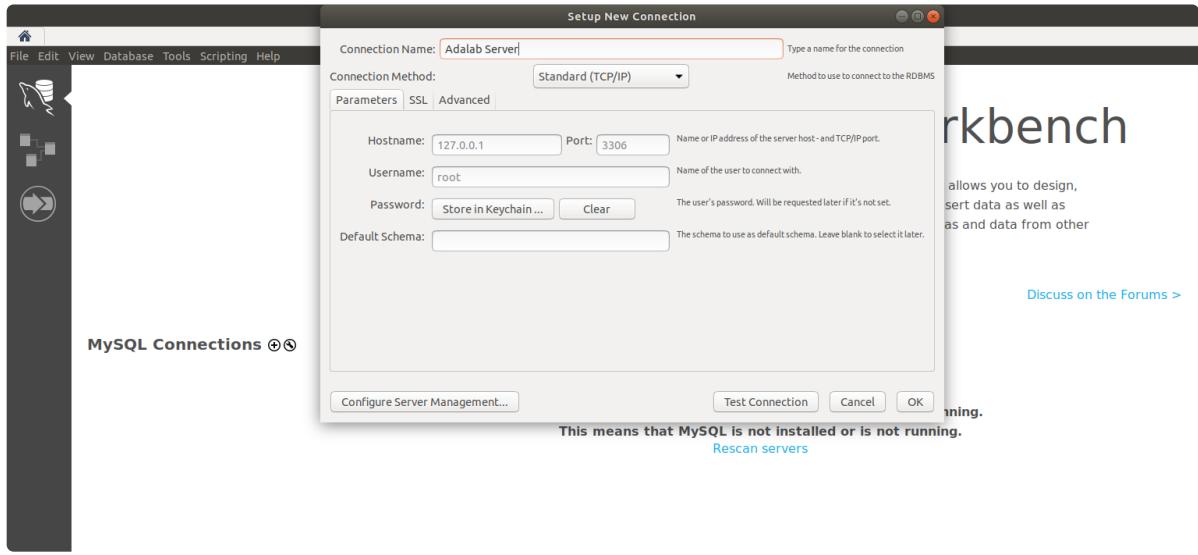
13. Ahora procedemos a abrir MySQL Workbench, buscándolo en la lista de aplicaciones.

14. Una vez abierto nos encontraremos con la siguiente ventana. Y le damos al símbolo de +.





15. Le damos ahora al símbolo de + y nos llevará a la siguiente ventana:

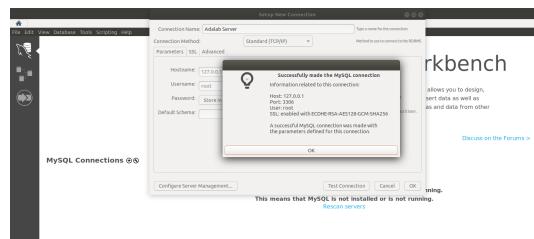


En las opciones nos aseguramos de que los siguientes campos están escritos de la siguiente forma:

- **Connection name:** Adalab Server.
- **Hostname:** 127.0.0.1
- **Port:** 3306
- **Username:** root

Una vez revisadas las opciones, hacemos click en el cuadro de **Test Connection**, donde se nos pedirá la contraseña para la usuaria **root** (recordamos que la contraseña es **AlumnaAdalab**).

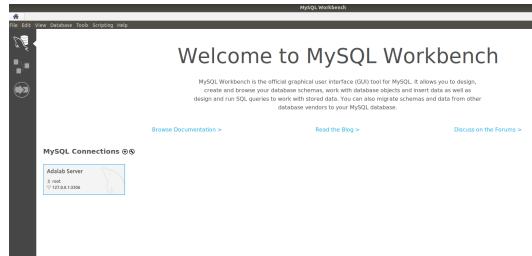
16. Si hemos ingresado los datos anteriores correctamente nos saldrá la siguiente ventana indicándonos de que se ha podido establecer la conexión con el servidor:



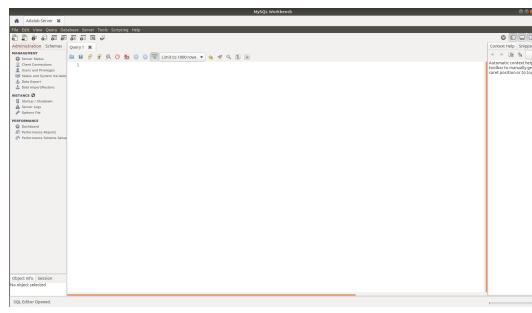
Clickeamos ahora en Ok en la ventana donde se nos ha informado de la conexión correcta al servidor y de nuevo a Ok, para guardar la configuración del servidor que hemos añadido a MySQL Workbench.

17. Si hemos realizado todo lo anterior correctamente, tendremos ante nosotras la siguiente

ventana:

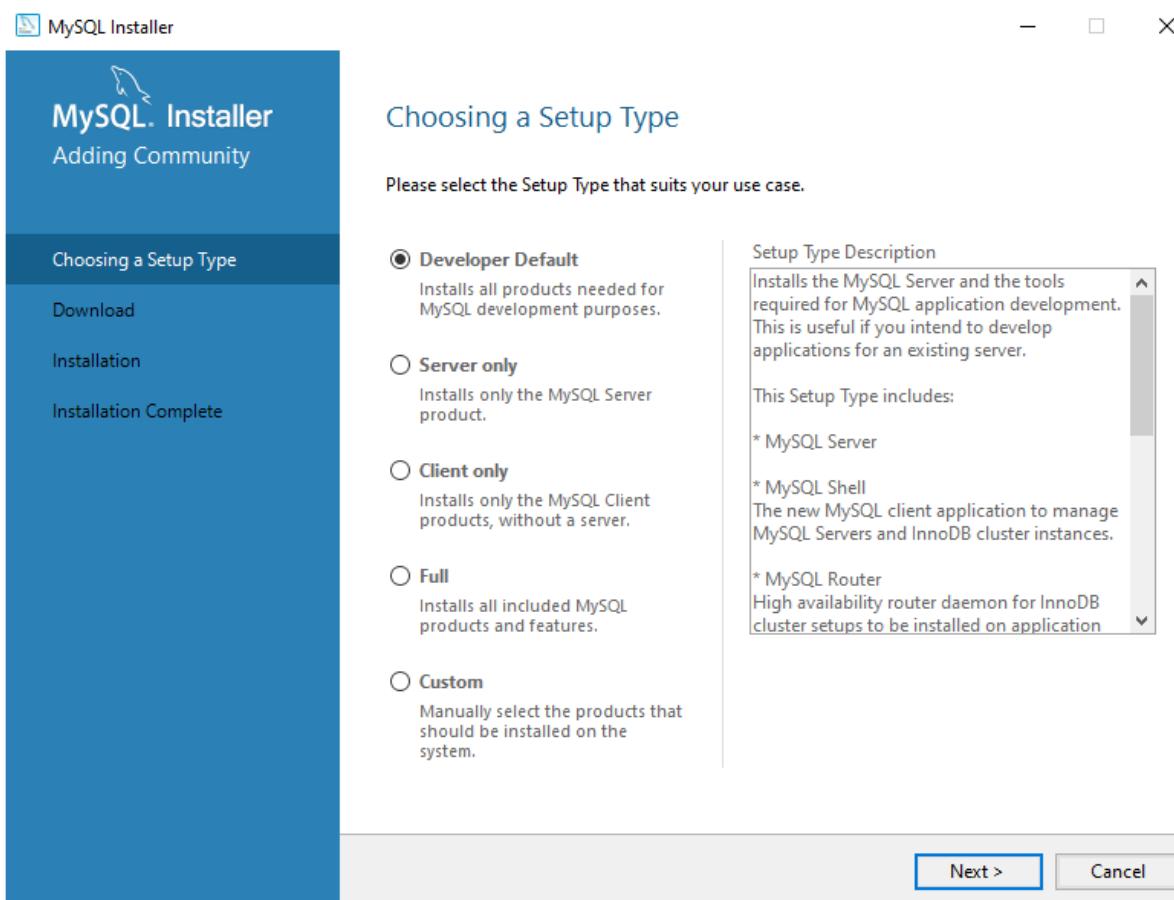


18. Clickeando ahora sobre el cuadro donde pone Adalab Server se nos llevará a la siguiente ventana en la cual ya podremos empezar a hacer uso de MySQL Workbench.

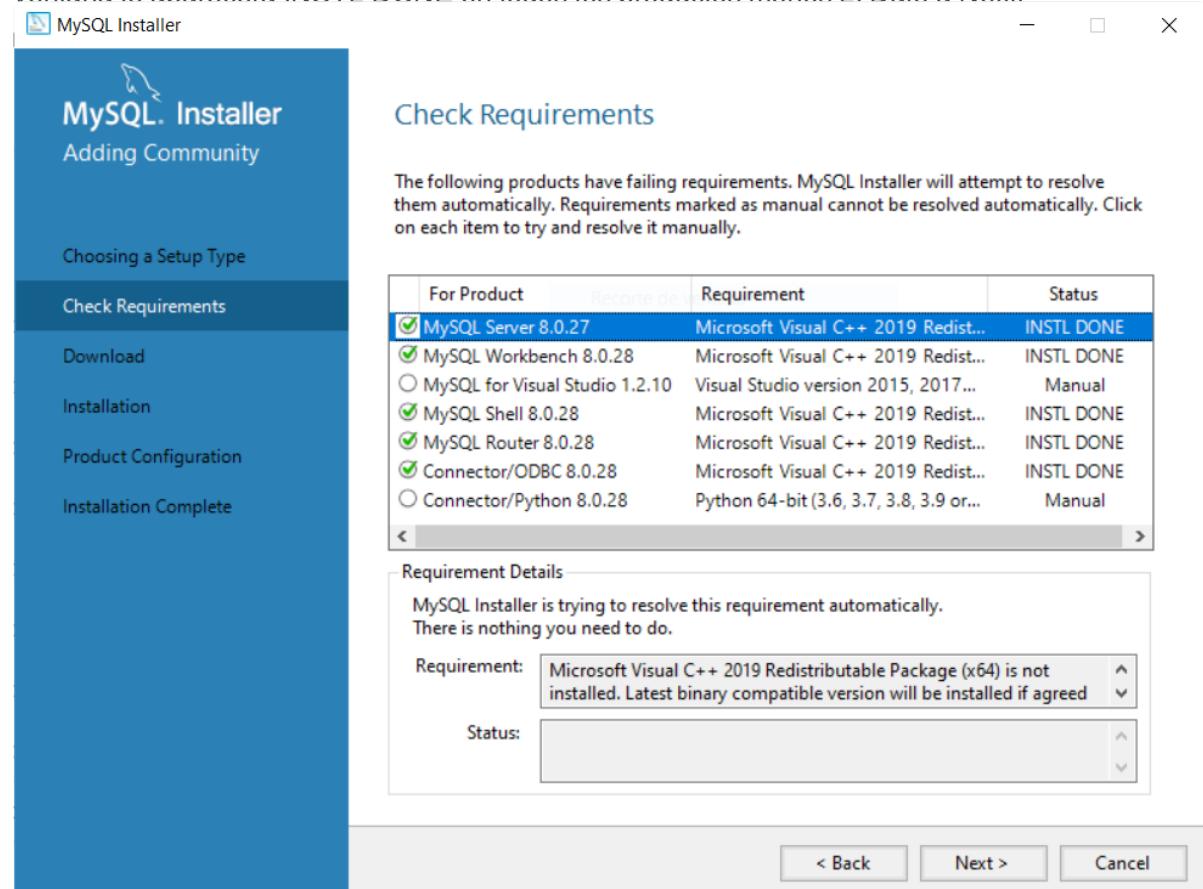


Windows 10

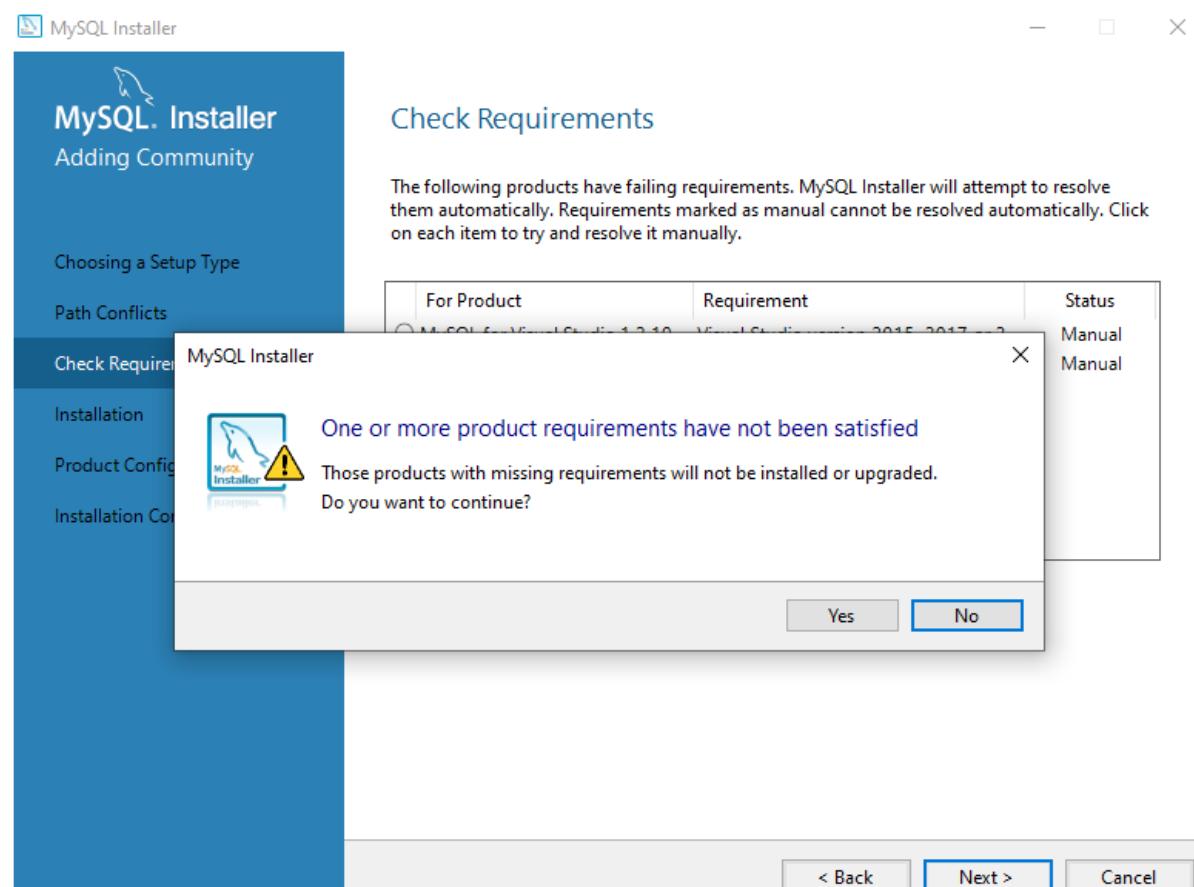
1. Descarga el [instalador MySQL](#). Y ejecútalo.
2. Por defecto debería estar seleccionada la opción **Developer default**, si no lo estuviera selecciónalo y dale a Next.



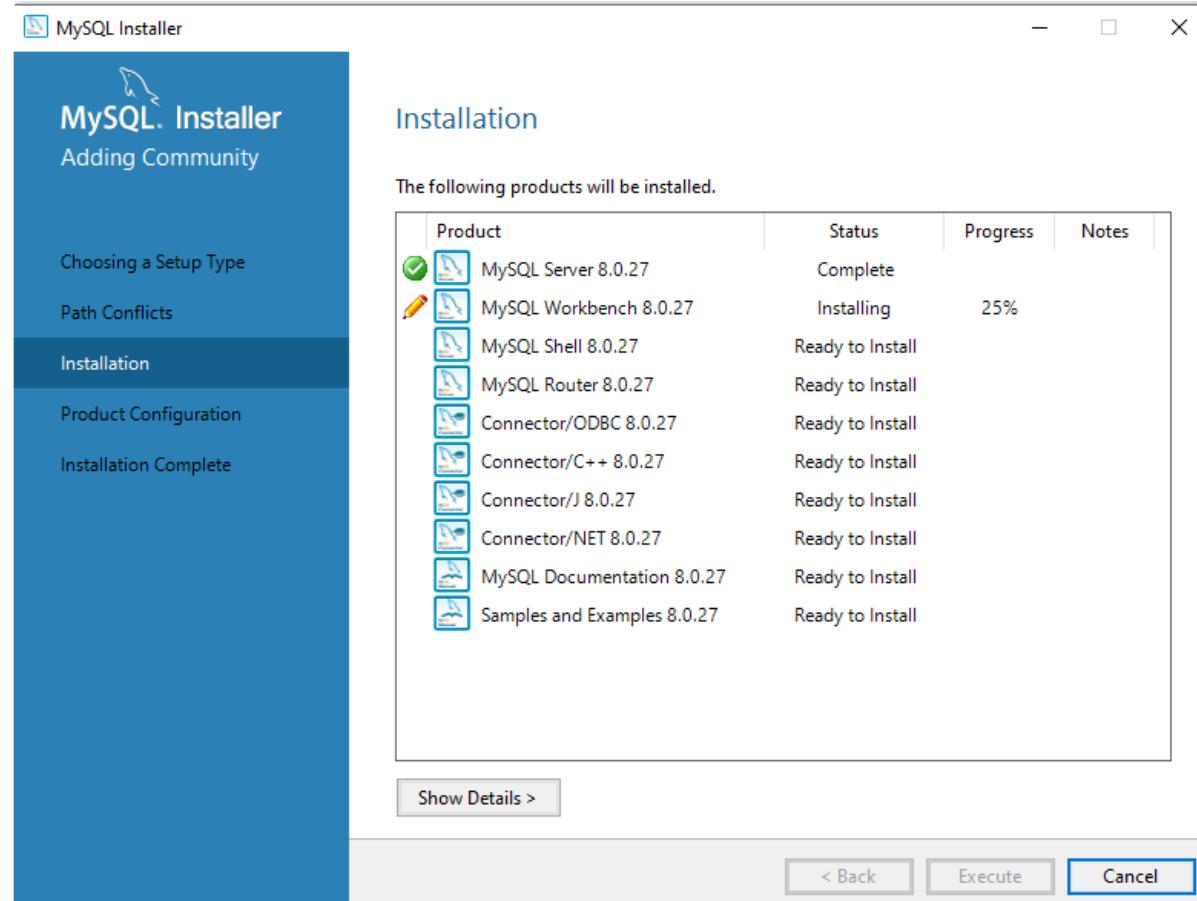
3. En la siguiente ventana puede que te aparece una lista que empiece con MySQL Server. Si es así, píñchalo y dale a Ejecutar abajo. Te abrirá otra ventana para instalar estas dependencias. Si no te aparece lo tendrás ya instalado por otra aplicación, en ese caso sólo te saldrán dos productos en este menú. Al haber terminado el proceso de instalación en la otra ventana te anárecerá INSTL DONE en todos los productos menos 2. Dale a Next.



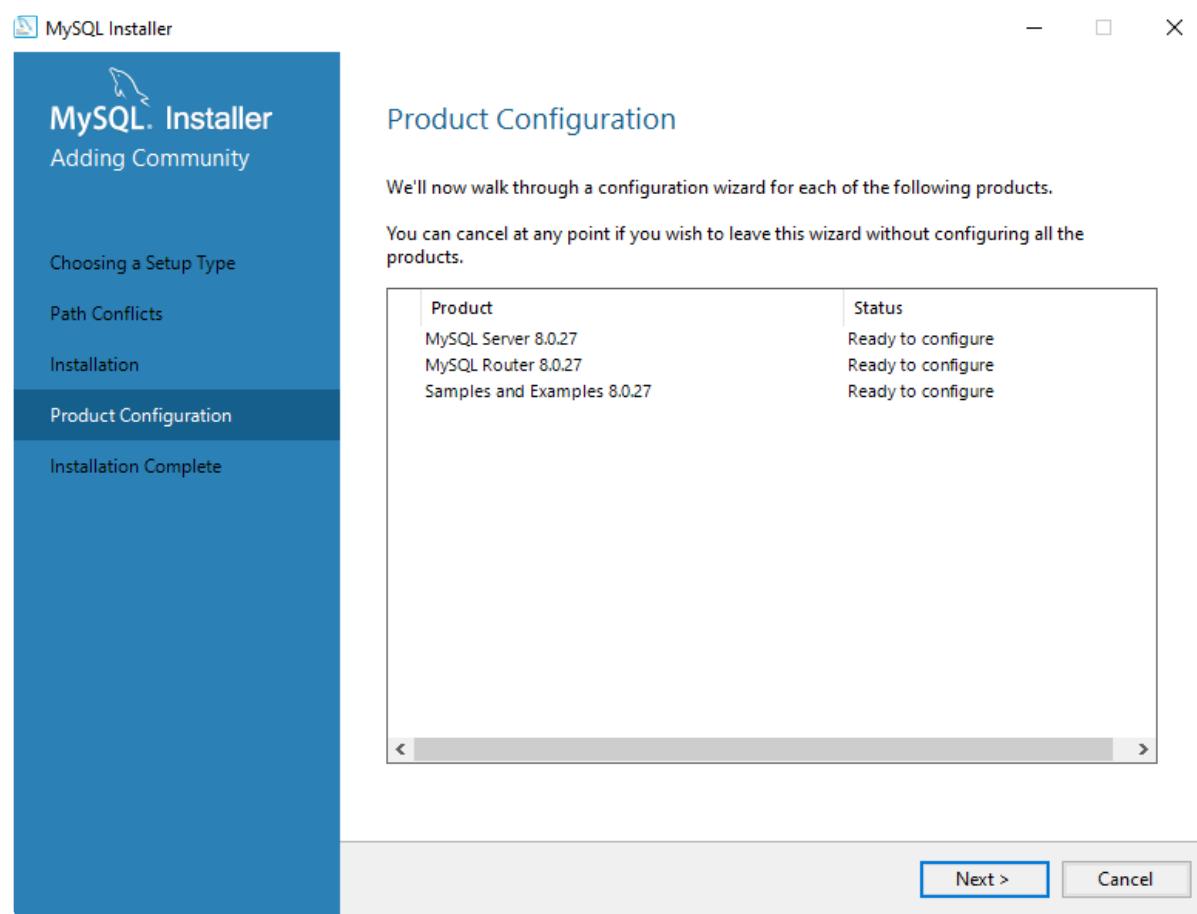
4. Te saldrá un warning por no encontrar Python instalado, ignóralo y dale a Yes.



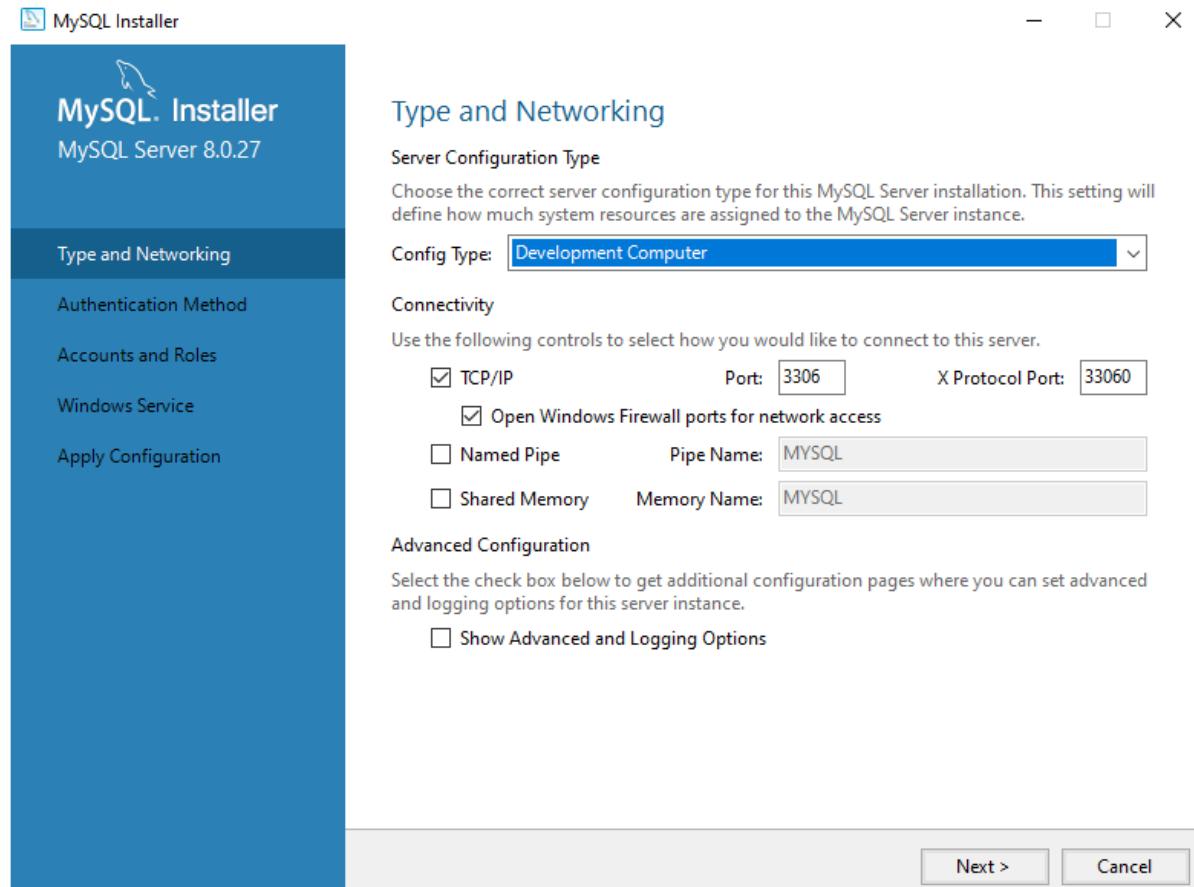
5. Dale a Ejecutar en la siguiente ventana. Comenzará el proceso de instalación y espera hasta que termine.



6. Una vez termine, empezaremos la configuración de los programas instalados.

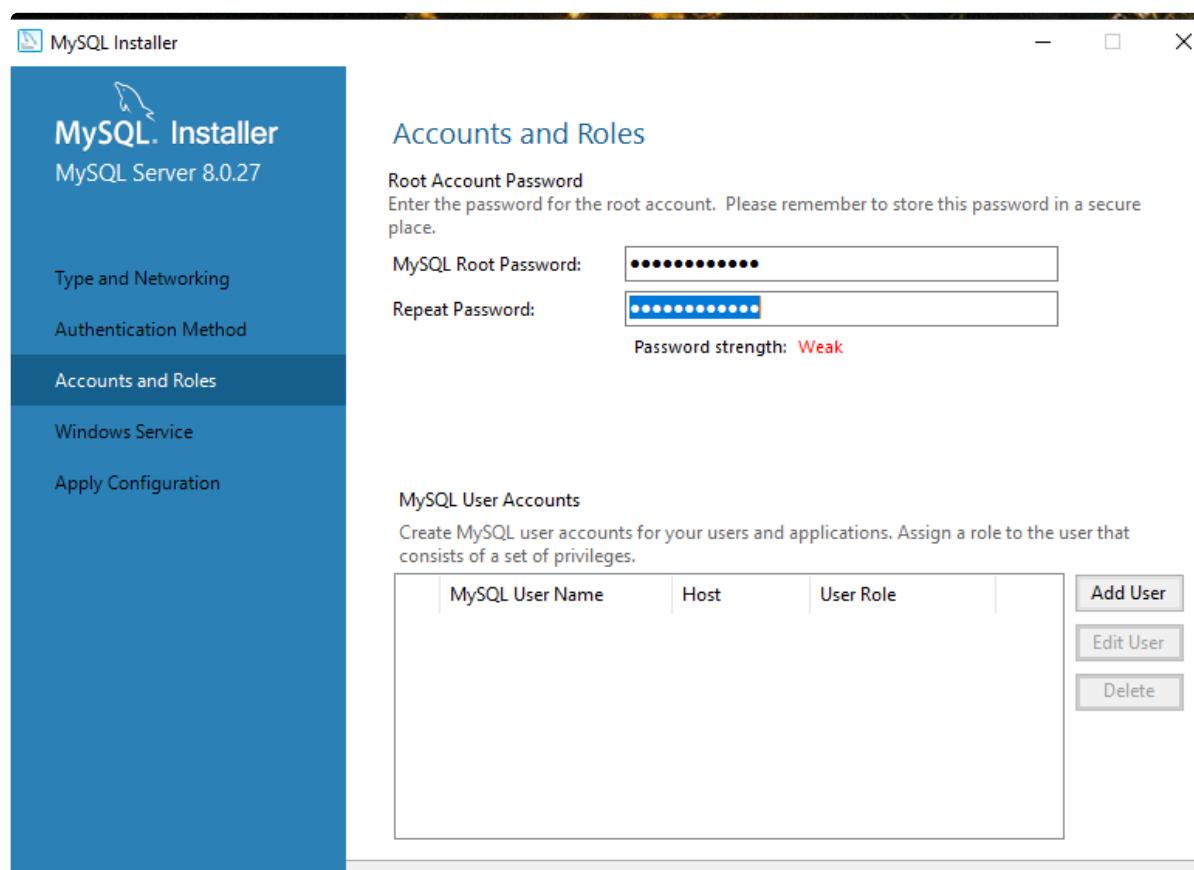


7. Por defecto en **Server configuration type** deberíamos tener seleccionado **Development Computer** si no seleccionalo y dale a Next



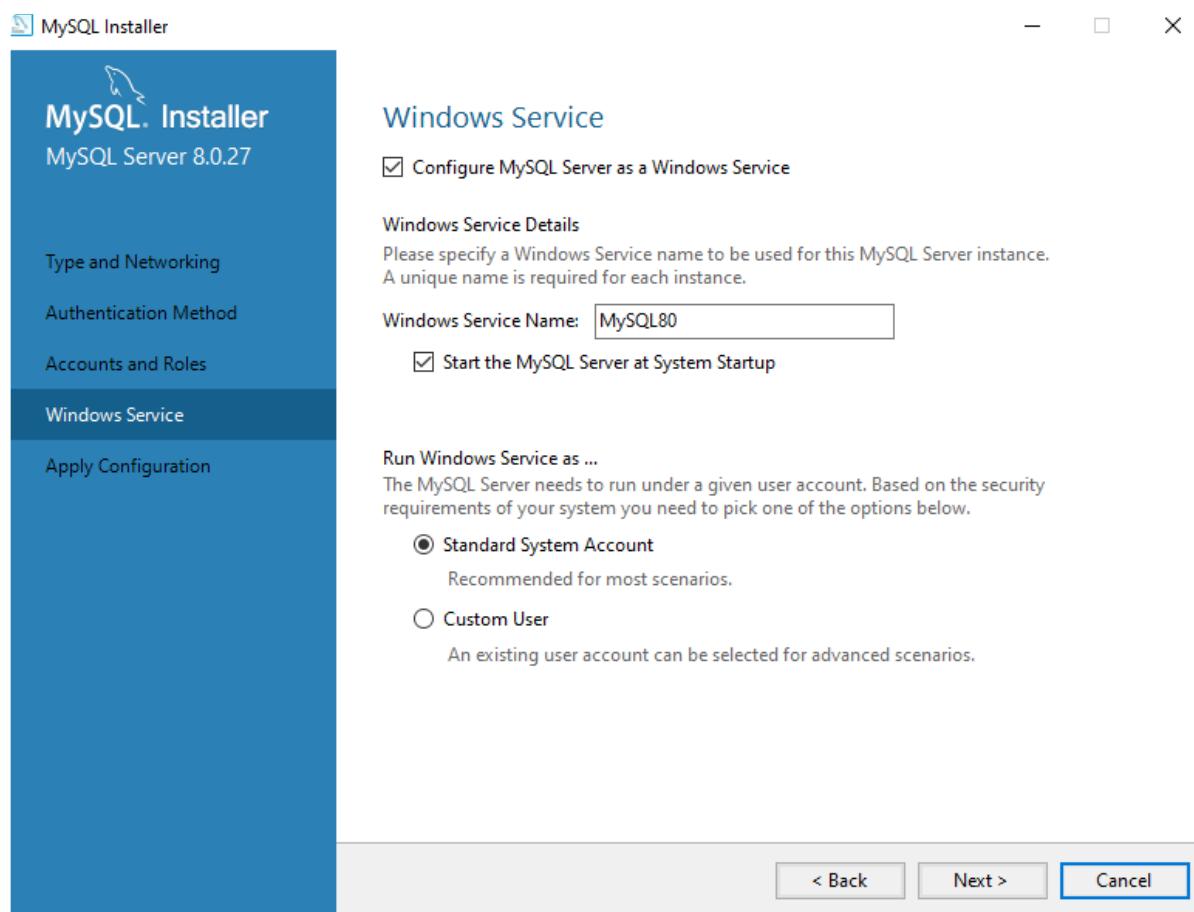
8. En la siguiente ventana, asegúrate de que esta seleccionado: **Use Strong Password Encryption for Authentication (RECOMMENDED)**, debería estar seleccionado por defecto.

9. Ahora en la ventana de **Accounts and Roles** introduce la contraseña para la usuaria **root**, usaremos **AlumnaAdalab**. Haz click en Next.



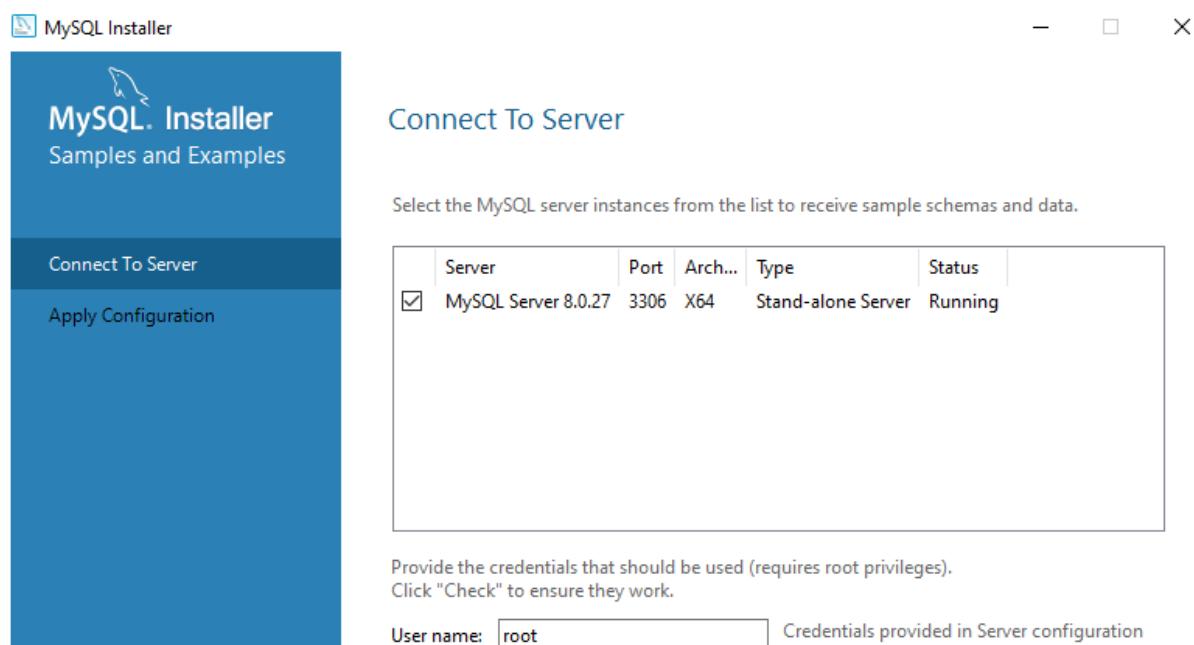
< Back Next > Cancel

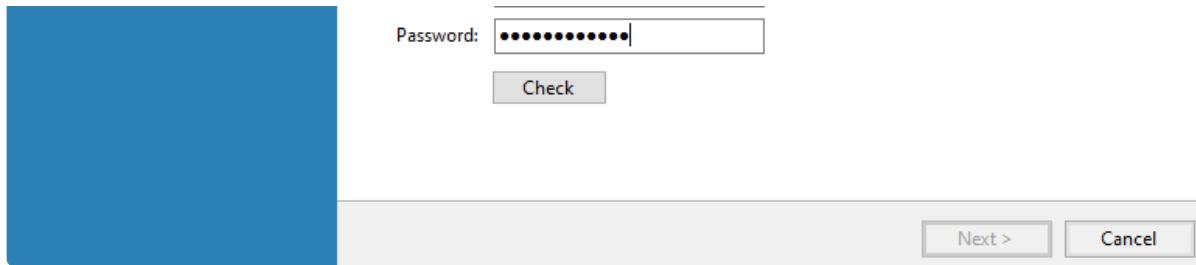
10. En la ventana de **Windows Service**, deja todas las opciones por defecto.



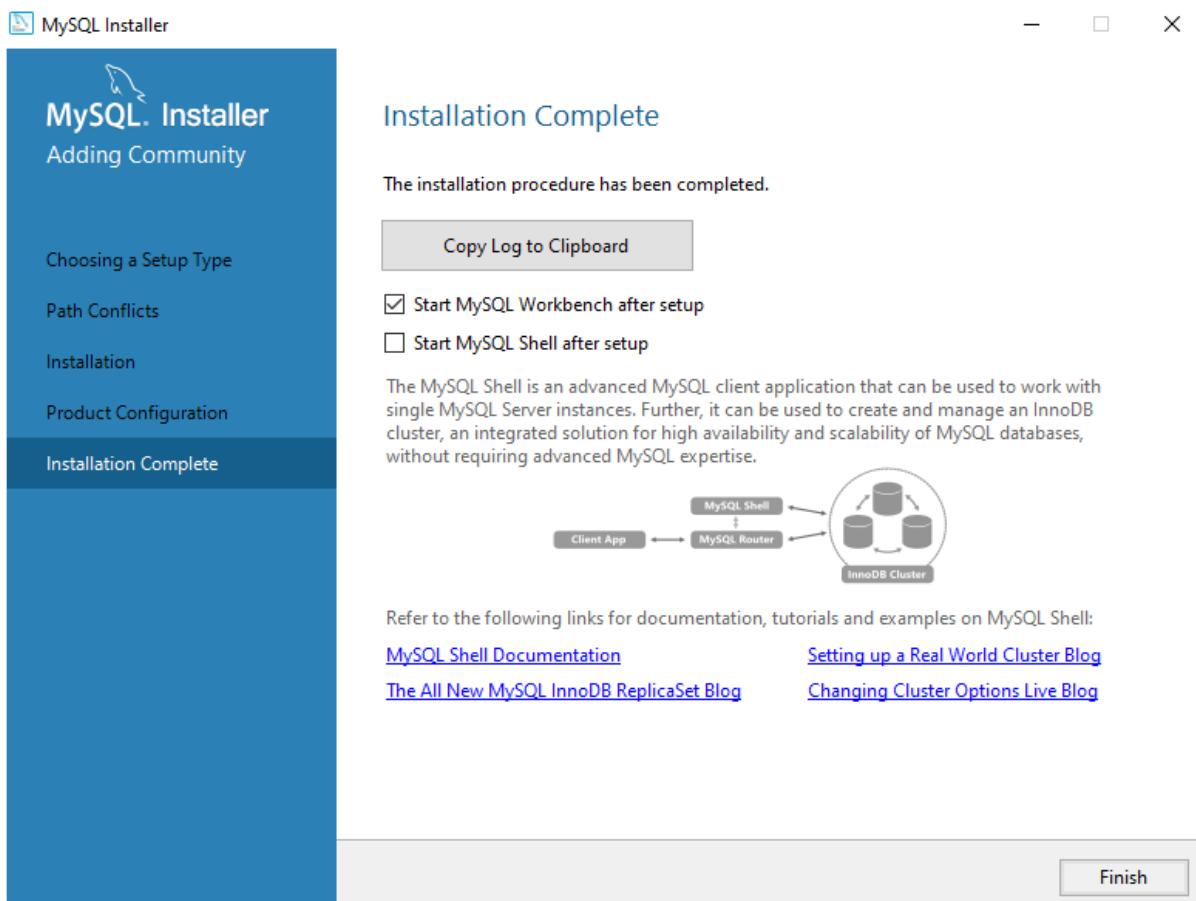
11. En la siguiente ventana haz click en Ejecutar para comenzar la configuración. Puede que aparezca una ventana adicional para configurar MySQL Router. De ser así deja todo por defecto y continúa con la instalación.

12. Cuando aparezca la ventana de **Connect to Server** (en la que vamos a conectarnos al servidor), asegúrate que como *User name* usamos **root** y que como *Password* usamos **AlumnaAdalab**, le damos a Check y si se conecta con éxito podemos hacer click en Next pasando así a una nueva ventana donde el programa de instalación aplicará la configuración que hemos establecido.

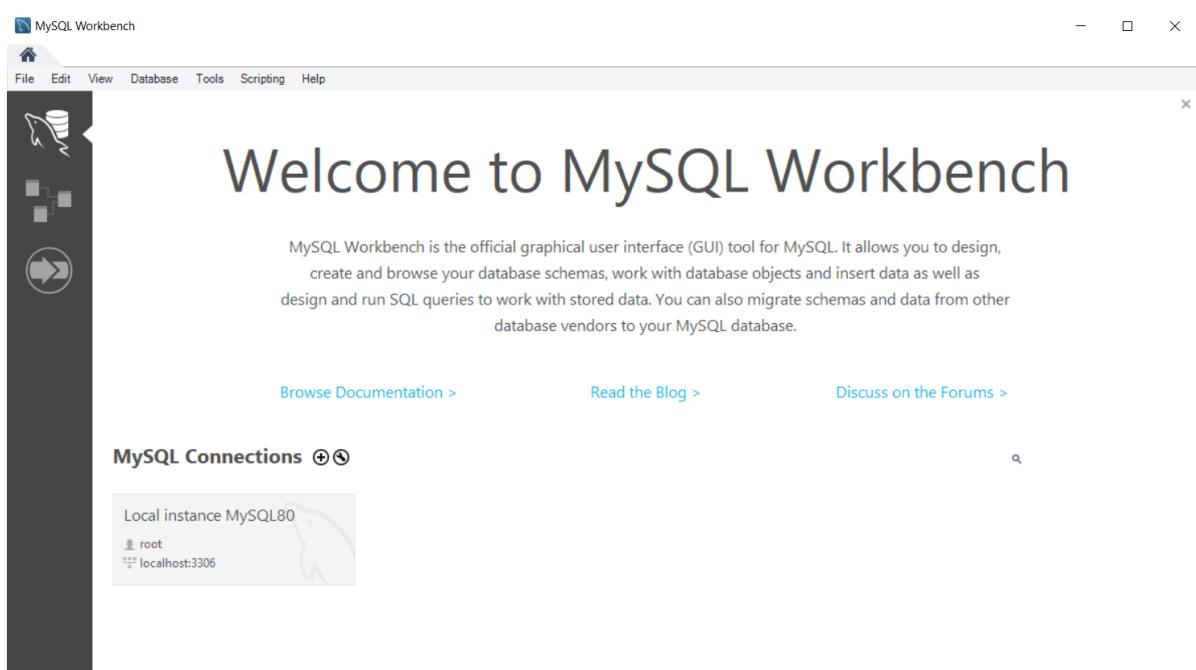




13. Tras esto ya habremos finalizado la instalación, podemos darle a Finish y se cerrará el instalador de MySQL.

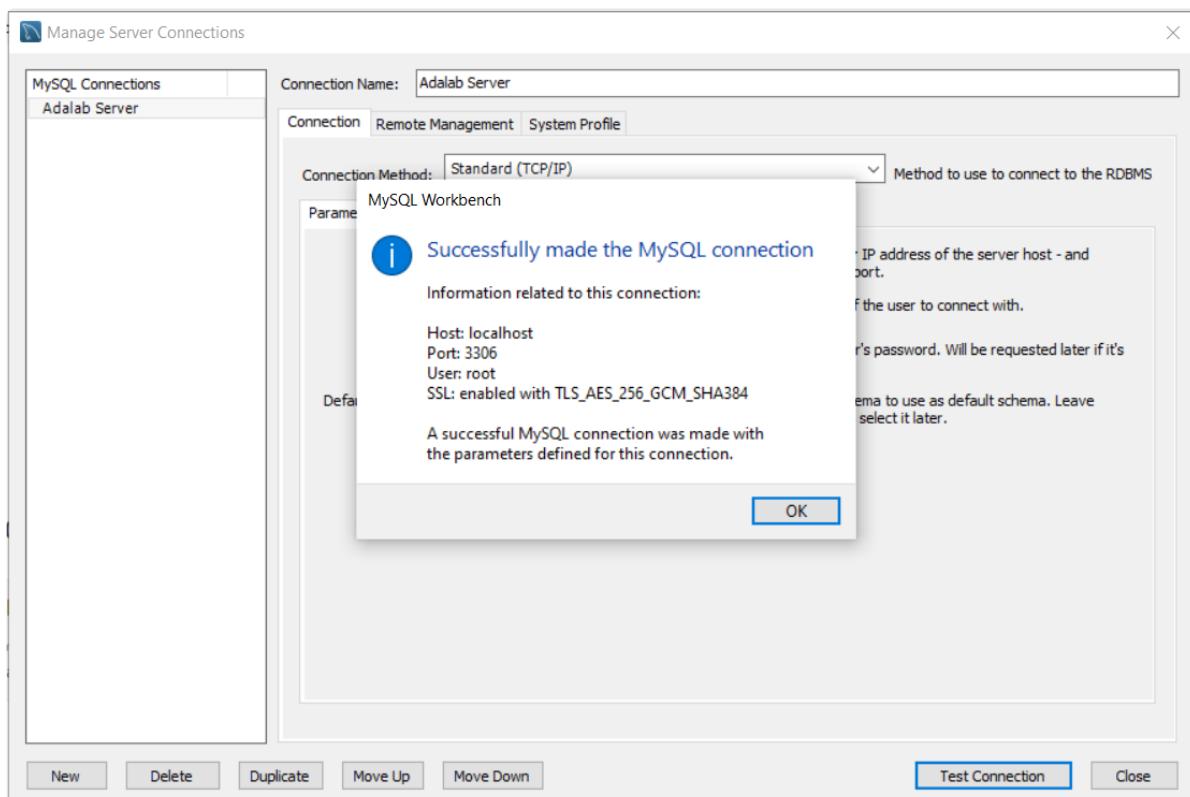


14. Al abrir el MySQL Workbench, te aparecerá una conexión a Local instance MySQL80. Al hacer click con el botón derecho podemos modificar esa conexión con Edit connection.



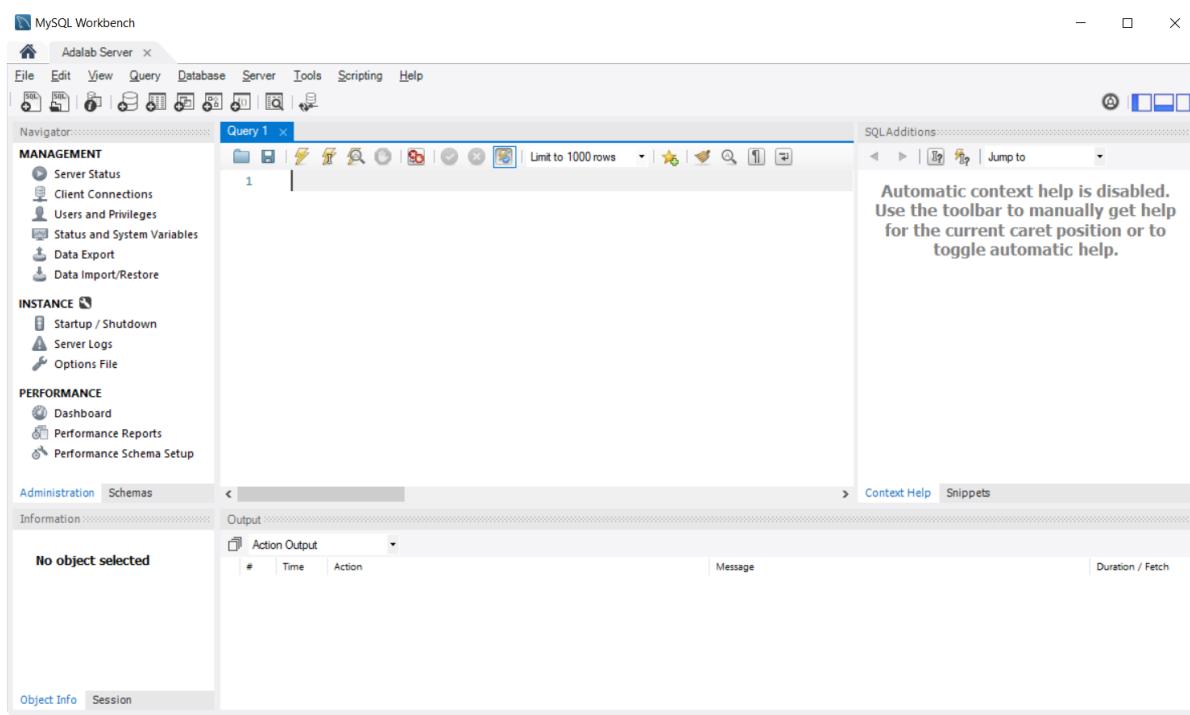


15. Cambia el nombre a **Adalab Server** y dale a Test Connection.



16. Ahora en el MySQL Workbench tendrás una conexión configurada llamada **Adalab Server**, si haces click sobre esa conexión el programa debería pedirte la contraseña de la usuaria **root** (recuerda **AlumnaAdalab**) y ya estaría todo configurado!

17. Al iniciar la conexión el programa debería llevarnos a la siguiente ventana donde ya podremos empezar a hacer uso de MySQL Workbench:



Enhorabuena, has finalizado la instalación de MySQL y MySQL Workbench!

Instalación de Tableau

Qué es Tableau?



Tableau es una plataforma de análisis visual que permite la elaboración de cuadros de mando, que permita analizar y reportar la información de forma visual.

Instalación

macOS

1. Descarga [Tableau public](#).
2. Abre el archivo descargado y instala la aplicación como harías normalmente y listo!

Ubuntu 18

1. Instalar Tableau en Linux es imposible, por lo tanto la profe encargada os explicará a aquellas alumnas una forma alternativa de poder utilizar Tableau en Linux. Esto se verá en el módulo 3. Por lo tanto no os preocupeis por ello ahora mismo :D

Windows 10

1. Descarga el [Tableau public](#).
2. Ahora instala el programa como lo harías normalmente y listo!

Enhorabuena, has finalizado la instalación de Tableau!

Introducción a las herramientas

Visual Studio Code

La paleta de comandos de Code

Code tiene una paleta de comandos que nos permite acceder a los comandos del editor mediante el teclado. La podemos abrir con `Ctrl+Shift+P` o `⌘+Shift+P` o la tecla F1.

El explorador de archivos en Code

Con `Ctrl+P` (Linux) y `⌘P` (macOs) podemos abrir el explorador de archivos de code, conforme vayamos escribiendo Code nos irá mostrando la coincidencias, y al seleccionar un archivo lo abrirá.

Indentando el código

Indentar es mover un bloque de código hacia la derecha, separándolo del margen izquierdo. No afecta al resultado final, si no a la legibilidad del código. A partir de ahora **vamos a indentar siempre** nuestro código, con la tecla *Tab*, conforme vayamos escribiéndolo, nos ayudará a que sea más legible y a prevenir e identificar errores más fácilmente.

- [Qué es la indentación](#)
- [Explicación de indentación en wikipedia](#)

Auto-indent en Code

Aunque vayamos indentando a la par que escribimos, a veces se nos queda alguna línea dónde no debe, o tenemos fallos que no vemos como una etiqueta no cerrada, para solucionarlo podemos apoyarnos en la funcionalidad auto-indent de Code.

- Con `Shift+Alt+F` (Windows), `Ctrl+Shift+I` (Ubuntu) o `⌃F` (MacOS) indentaremos todo el documento

Cómo organizar nuestro proyecto

A la hora de organizar los archivos y carpetas de un proyecto es normal fijar unas pequeñas normas que pueda seguir todo el equipo de manera que no sea un caos de archivos y cualquier persona pueda orientarse rápidamente en el proyecto y/o seguirlo.

No hay una manera "buena" y cada empresa tiene las suyas. Lo importante es tener unas normas que las personas del equipo (y aquellas que se unirán en un futuro) puedan seguir facilitando su día a día. Para este curso vamos a plantear unas que dan bastante buen resultado para empezar. Aquí van:

- Los nombres de archivos y carpetas irán siempre en minúsculas, sin tildes, sin espacios y sin caracteres especiales.
- Usaremos guiones para separar palabras: `data-processed.csv`
- Usaremos [rutas relativas](#) siempre. Veremos más información sobre las rutas relativas en la siguiente sesión.
- Los nombres de archivo siempre en inglés y deben ser lo más concisos posibles: `project-name.ipynb`.
- Se crearán carpetas para guardar los datos de entrada y de salida.
 - Carpeta input: Para almacenar los datos iniciales.
 - Carpeta output: Para almacenar los datos procesados.

Para la estructura del proyecto usaremos estas carpetas:

```
project-folder
  └── input
    └── data.csv
  └── output
    └── data-processed.csv
  └── project-name.ipynb
```

Atajos de teclado

Hoy hemos visto un montón de atajos de teclado, en este momento puede que sientas un poco de resistencia a utilizarlos porque:

- Te sientes más cómoda y rápida con el trackpad o ratón, ¡es normal! Llevas toda la vida acostumbrándote a usarlo a fuerza de repetición. Así que vamos a cambiar esto poco a poco, intentando usar los atajos en la medida de lo posible, con el tiempo lo harás sin pensar y serás mucho más ágil utilizando el ordenador.
- Que estás echando un lio con tanto comando, ¿para qué era cada uno? Vamos a tener a mano un documento que nos ayude a recordar.

En Code:

- [Atajos de teclado de Visual Studio Code para Windows](#)
- [Atajos de teclado de Visual Studio Code para Linux](#)
- [Atajos de teclado de Visual Studio Code para macOS](#)

Jupyter Notebook

Jupyter Notebook es un entorno de trabajo basado en un navegador web que permite trabajar siguiendo el formato de celdas. Donde podemos escribir código, texto o fórmulas matemáticas, entre otros. Esto nos permite ejecutar código haciendo uso de un navegador web (siempre que se haya instalado en el equipo). En nuestro caso al haber instalado Anaconda, nos viene por defecto configurado para poder ejecutar código de Python haciendo uso de esta herramienta.

¿Por qué usaremos Jupyter notebook?

La versatilidad de los Jupyter Notebooks reside en que podemos incluir celdas con código, texto, imágenes, entre otros. Así mismo, dentro de los notebooks cada celda se puede ejecutar de forma independiente a la anterior, por lo que podremos ir ejecutando el código a medida que lo vayamos desarrollando para comprobar que funciona como se espera. Así mismo, nos permitirá ver los conjuntos de datos como si estuviéramos viendo una especie de 'excel' dentro del propio notebook, exportar y guardar aquellas gráficas que se realicen, etc.

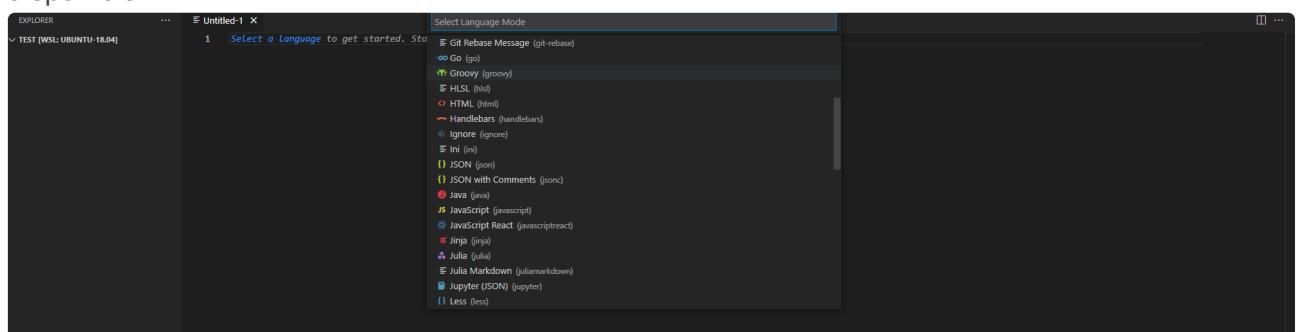
Adicionalmente, VS Code permite utilizar los notebooks como si fuera un explorador web, lo que nos facilita poder utilizar los plugins y trabajar con un control de versiones como con **Git/GitHub**.

¡Como se puede entrever el potencial de uso de los Jupyter notebooks es bastante amplio!

Creando un Jupyter notebook

Para crear un nuevo Jupyter notebook podemos utilizar dos procedimientos similares:

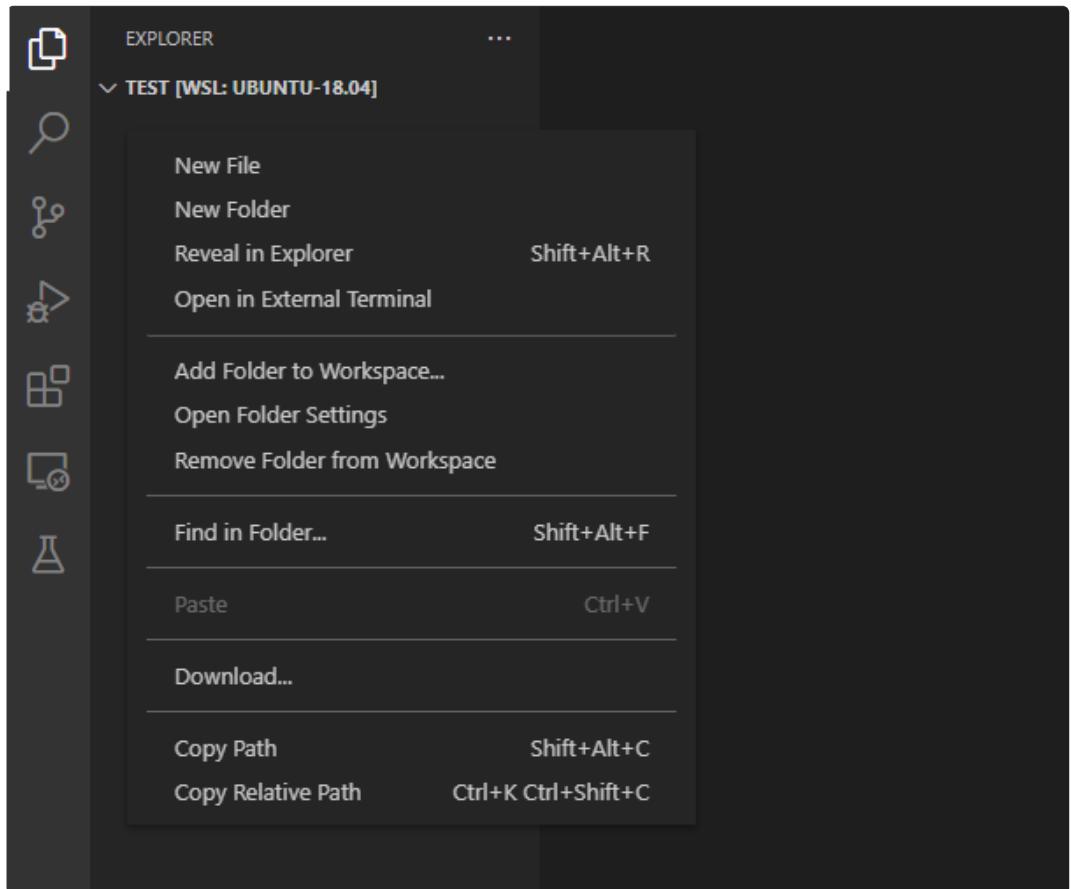
- Ir a `File(Archivo)`, `New file(Archivo nuevo)` y seleccionar Jupyter Notebook de la lista disponible.



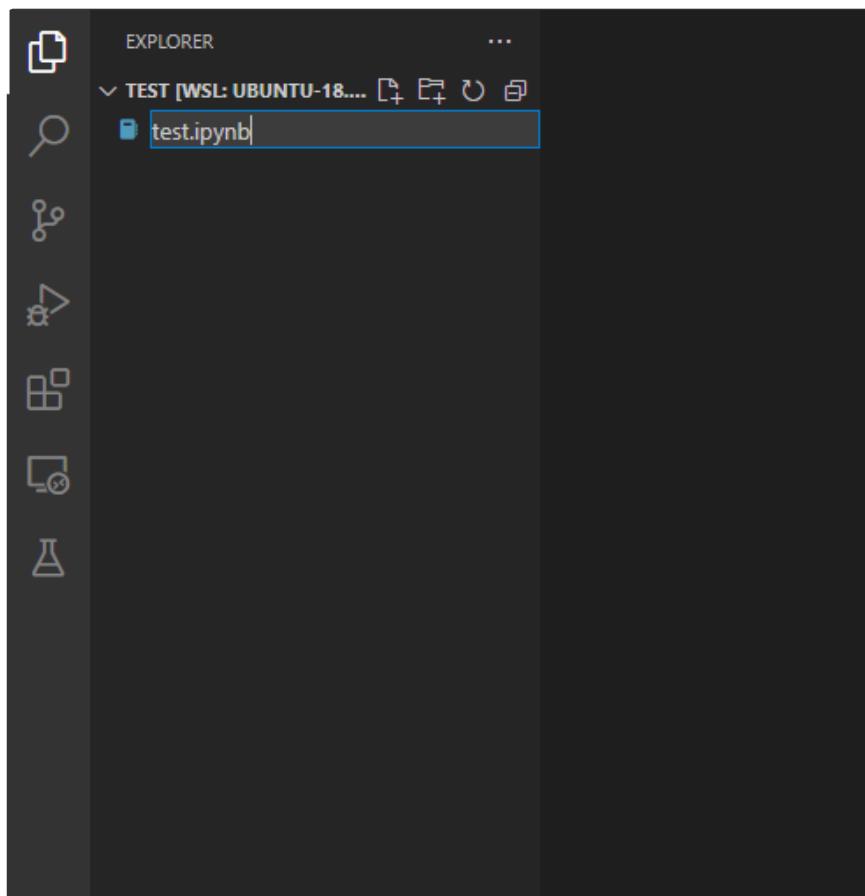
Listado de lenguajes de programación VS Code

Guardar el archivo nuevo, cerrar y volver a abrirlo.

- Click secundario del ratón dentro de la zona de explorador del VS Code, `New file(Archivo nuevo)` y guardarlo con la extensión `.ipynb`.

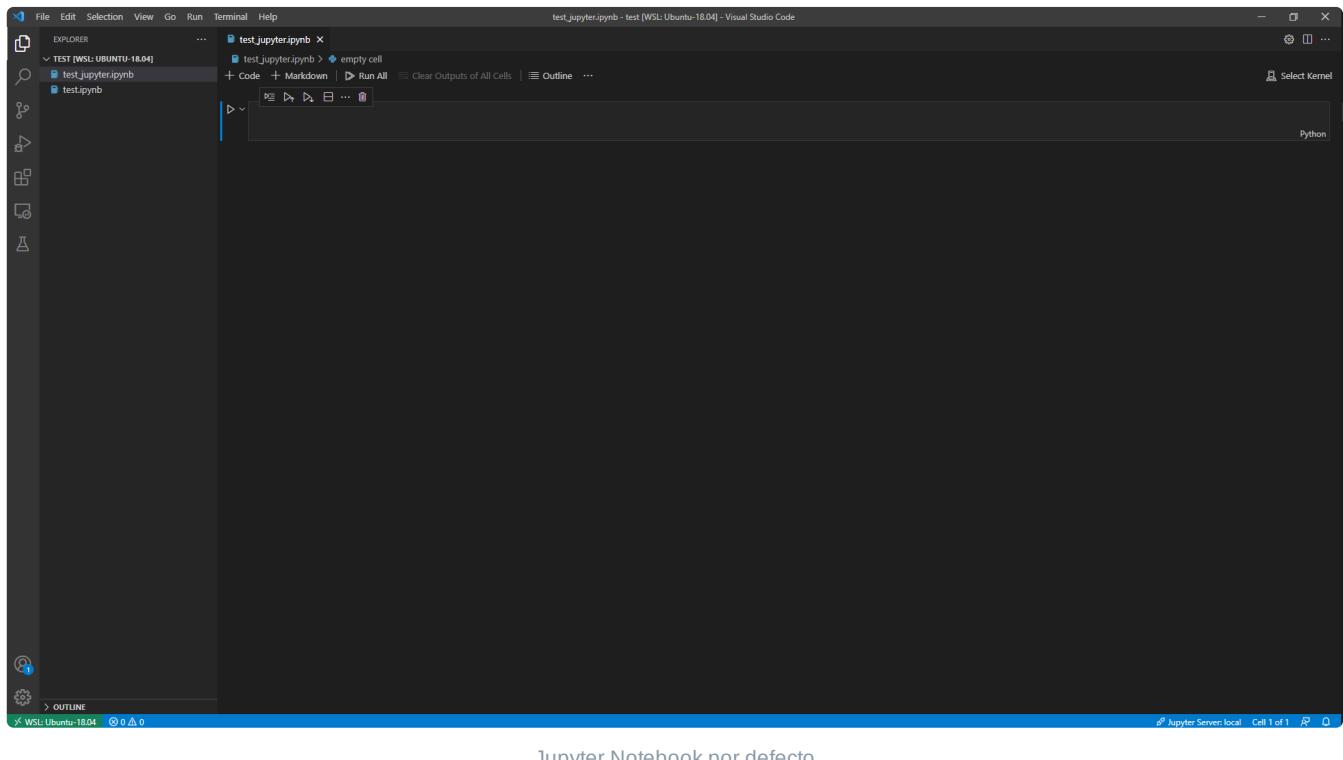


Creación de archivo nuevo



Nombre del archivo y extensión

Podemos ver en la siguiente imagen como queda el nuevo notebook por defecto:



Jupyter Notebook por defecto

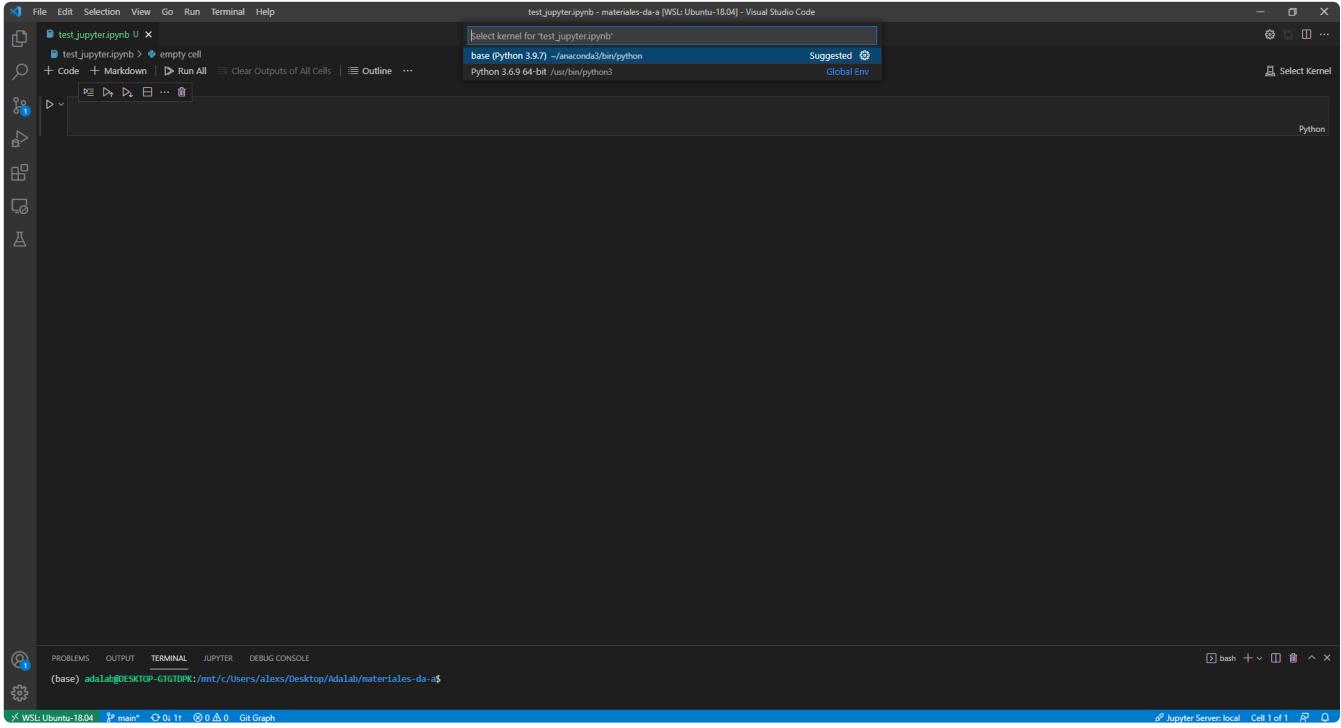
Seleccionando el kernel de Jupyter notebook

Ahora que ya hemos creado un nuevo Jupyter notebook, nos quedará indicarle al notebook el kernel (en nuestro caso, esto será el interprete de Python que utilizará el Jupyter Notebook), ya que si no, no seremos capaces de ejecutar código de Python. Si te fijas en la esquina superior derecha de la imagen anterior, podemos ver que tenemos `Select kernel`. Para seleccionar el kernel seguimos los siguientes pasos:

1. Clicamos sobre el

`Select Kernel`

2. Seleccionamos de la lista desplegable el kernel con nombre `base` que esta en la carpeta `~/Anaconda/bin/python`. (Si no aparece y trabajas en Windows seguramente no tendrás la extensión [instalada para WSL](#).)



3. Si se ha realizado correctamente el proceso ahora habrá cambiado de

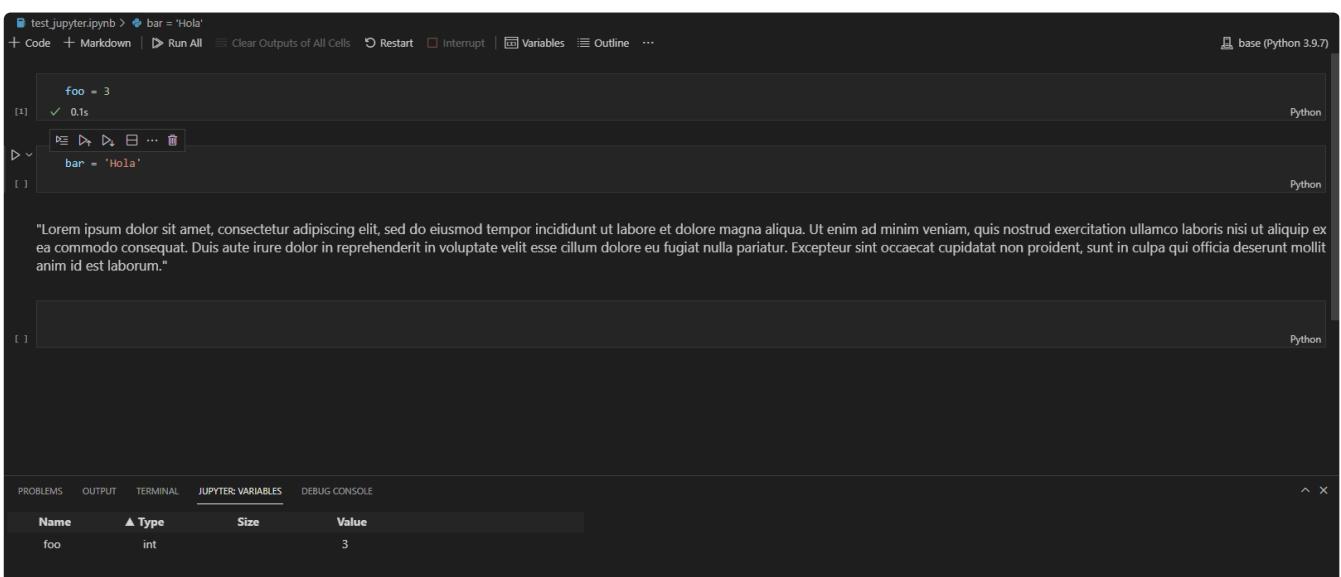
 a



Usando un Jupyter notebook

Llegados a este punto ya podemos comenzar a utilizar Jupyter notebook para ejecutar código Python!

Vamos a explicar brevemente las partes y opciones que componen un Jupyter notebook.



Jupyter Notebook con algunas celdas

Vamos a listar las siguientes partes que componen el Notebook y lo que hacen:

- Crea una nueva celda de código.
- Crea una nueva celda en modo markdown (texto).
- Para ejecutar todas las celdas del Notebook.
- Limpia todas las salidas de las celdas que se han ejecutado.
- Reinicia el Jupyter notebook y sus salidas.
- Para parar la ejecución del Notebook. Utilizado si deseamos terminar la ejecución de una celda, ya sea por que esta tardando mucho o porque sabemos que ha explotado y no esta funcionando correctamente.
- Nos muestra en otra ventana de VS Code las variables existentes (ejecutadas).
- Nos ejecuta la celda, si hay código lo ejecuta también. Si es una celda de tipo markdown, formatea el texto para no ver el cuadro de la propia celda.
- Ejecuta las celdas por encima de la seleccionada.
- Ejecuta las celdas por debajo de la seleccionada.
- Nos parte una celda en dos, siempre que tengamos más de una línea por celda, en función de la posición del cursor.
- Nos elimina la celda que tengamos seleccionada.

Ahora vamos a explicar brevemente algunas de las características de las celdas y su contenido.

- Primera celda: Contiene la siguiente variable `foo = 3`, podemos comprobar que se ha ejecutado, ya que nos sale un tick verde y su tiempo de ejecución. Adicionalmente se ve debajo del botón de ejecutar celda [1] que indica el orden del número de celda que se ha ejecutado.

- Segunda celda: Contiene la siguiente variable `bar = 'Hola'`. No se ha ejecutado ya que tenemos debajo del botón de ejecutar celda [] .

- Tercera celda: Celda de tipo Markdown, que contiene únicamente texto y que se ha ejecutado. Ya que

no podemos apreciar los bordes de la celda.

```
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum."
```

- Cuarta celda: Celda vacía, en la que aún no se ha introducido nada.

Atajos de teclado para los Jupyter Notebooks en VS Code.

Para utilizar estos atajos siempre tenemos que haber seleccionado la celda en la que queremos realizar la acción del atajo del teclado. Haciendo click en la zona de selección de celda.



- Presionando la tecla `A`, creamos una nueva celda de tipo código encima de la seleccionada.
- Presionando la tecla `B`, creamos una nueva celda de tipo código debajo de la seleccionada.
- Presionando la tecla `Y`, convertimos una celda de tipo markdown a tipo código.
- Presionando la tecla `M`, convertimos una celda de tipo código a tipo texto.
- Presionando `D D` (dos veces seguidas la tecla), eliminamos la celda con la que estamos trabajando.
Recomendamos no utilizarlo y borrar las celdas usando el botón para evitar meteduras de pata
- Presionando `Ctrl + Enter`, ejecutamos la celda seleccionada.

A practicar

Durante los primeros días del curso te explicaremos en detalle cómo se usan los Jupyter notebooks y sus comandos. Pero es interesante que empieces a familiarizarte un poco, así que te recomendamos que hagas el siguiente ejercicio:

1. Crea un nuevo Jupyter Notebook. Utilizando ambos métodos explicados.
2. Selecciona el kernel a utilizar.
3. Crea algunas celdas de tipo normal.
4. Crea algunas celdas de tipo markdown.
5. Trastea y prueba cosas...

La mejor forma de aprender una nueva herramienta es jugar con ella, trastea sin miedo, rompe cosas y si aparecen errores no te preocupes ¡Las profesoras estaremos encantadas de ayudarte!

Terminal

Como ya hemos instalado y configurado la terminal de nuestro ordenador vamos a explicar para qué es y cómo se usa. Si te resulta complejo no te preocupes es algo que vamos a usar y practicar durante todo el curso. ¡¡¡Te ayudaremos a dominarla!!!

¿Por qué necesitamos usar la terminal?

Como hemos comentado, la terminal, también llamada consola de comandos o shell, es una herramienta fundamental para la programación. Su finalidad es ejecutar comandos u órdenes mediante instrucciones. Estos comandos son similares a las interacciones que haríamos en una aplicación normal (clics, escribir en campos de texto, cambiar de sección, etc.) pero en este caso se hacen escribiendo órdenes en una terminal de comandos.

Muchas de las herramientas para programación están hechas sin interfaz porque son tan sencillas que no merece la pena hacer una interfaz o son tan complejas que no se puede hacer una interfaz gráfica que tenga todas las funcionalidades posibles. La solución es usar la terminal.

Cuando una persona utiliza un programa como Spotify o Chrome está utilizando una interfaz gráfica que transforma nuestras acciones en órdenes que le envía al sistema operativo a través de una terminal. Es decir las aplicaciones son intermediarias. Cuando utilizamos la terminal directamente no hay nada que se interponga entre el sistema operativo y nosotras. Tenemos todo el poder, y toda la responsabilidad claro, pero tranquila, no podemos romper nada

Partes de la terminal

En la imagen vemos una terminal abierta desde el programa VS Code. Vamos a explicar con este ejemplo las partes de la terminal. Es el momento de que abras una terminal dentro de tu VS Code y veas la información que te muestra.

```
migueldelmazo@Miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$ ls -la
total 32
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 09:56 .
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Dec  1 14:06 ..
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 10:52 .
-rw-rw-rwx 1 migueldelmazo migueldelmazo   35 Oct 10 2019 .gitignore
-rw-rw-rwx 1 migueldelmazo migueldelmazo 19045 Oct 10 2019 LICENSE
-rw-rw-rwx 1 migueldelmazo migueldelmazo 2805 Nov 18 10:40 README.md
-rw-rw-rwx 1 migueldelmazo migueldelmazo 5038 Nov 20 10:24 SUMMARY.md
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 assets
-rw-rw-rwx 1 migueldelmazo migueldelmazo 306 Apr 25 2020 book.json
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 10:23 deprecated
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 10:15 guias
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 instalacion
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 10:23 modulo_1
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 modulo_2
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 modulo_3
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 09:53 modulo_5
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 20 10:08 modulo_6
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 plantillas
drwxrwxrwx 1 migueldelmazo migueldelmazo 4096 Nov 18 10:37 proyectos
-rw-rw-rwx 1 migueldelmazo migueldelmazo 26 Jul 17 11:34 robots.txt
migueldelmazo@Miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$
```

Terminal en VS Code

Lo primero que nos muestra la terminal es el **prompt**, a partir del cual podemos escribir nuestros comandos. El prompt está compuesto por:

- Nombre del usuario con el que hemos iniciado sesión en el ordenador. En la imagen es **migueldelmazo**.
- @
- Nombre del equipo u ordenador. En la imagen es **Miguel**.
- Ruta de la carpeta en la que está ahora mismo la terminal:
 - Puede ser la ruta absoluta de una carpeta. En la imagen es `/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1`.
 - O podría ser otra cosa como `~`, que es una abreviatura de la carpeta **home** del ordenador de la usuaria.
- Por último el símbolo del dólar, que es simplemente para saber dónde termina el prompt y dónde podemos empezar a escribir un comando.

Nota: A veces, en nuestros materiales o en Internet, encontraremos ejemplos de comandos precedidos por el símbolo del dólar, por ejemplo `$ ls -la`. Es una manera de decir que es un comando de terminal. Nosotras **no** debemos escribir el `$` en la consola.

En el ejemplo también hemos escrito el comando `ls -la` y le hemos dado a intro. Este comando es para listar los ficheros y carpetas que están dentro de la carpeta en la que está la terminal. Es decir los ficheros y carpetas que están dentro de `/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-l`. Por ello la terminal nos muestra el listado de ficheros y carpetas que vemos en la imagen, con información relativa a cada uno de ellos.

Y ahora que ya sabemos cómo escribir comandos en la terminal vamos a ver los más comunes y usados en programación:

Comandos (más usados) de la terminal

ls (list)

El comando `ls` nos muestra un listado de los archivos y carpetas que hay en la carpeta actual.

```
ls
```

A los comandos se les puede pasar opciones. Podemos usar la opción especial `-la` para listar también los ficheros y carpetas ocultos. La opción `-l` indica que queremos ver los ficheros en modo listado. La opción `-a` indica que queremos ver todos (all) los ficheros y carpetas, incluidos los ocultos. La opción `-la` es la suma de las dos opciones juntas. Los ficheros y carpetas ocultos empiezan por `.` y por defecto no se ven `:)`. Por ejemplo un fichero oculto es `.gitignore`.

```
ls -la
```

Nota: Por cierto, ni Windows, ni Mac, ni Ubuntu muestran por defecto los ficheros ocultos de una carpeta. Pero si abrimos la carpeta desde VS Code, este sí muestra los ficheros ocultos. Sí quieres ver los ficheros y carpetas ocultos abre una carpeta en VS Code y pulsa en ícono de arriba a la izquierda  , aparecerán todos los ficheros que tenga esa carpeta.

cd (change directory)

El comando `cd` nos ofrece diferentes posibilidades a la hora de cambiar o movernos de carpeta. Para entrar en una carpeta hija de la carpeta actual usamos:

```
cd nombre-de-carpeta-hija
```

Podemos encadenar varios nombres de subcarpetas separadas por `/` para llegar hasta una ruta más profunda:

```
cd nombre-de-carpeta-hija/carpeta-nieta/carpeta-bisnieta
```

Los dos puntos `..` nos permite subir a la carpeta madre, esto es, ir a la carpeta que contiene nuestra carpeta actual:

```
cd ..
```

Nota: Todo lo que aprendimos sobre [rutas relativas y absolutas](#) lo usamos mucho con los comandos de la terminal, en especial con el con el comando `cd .`

mkdir (make directory)

Nos permite crear una carpeta. Pero no entra en la carpeta, solo la crea.

Para crear la carpeta `proyecto` escribimos el comando:

```
mkdir proyecto
```

Para entrar dentro de la carpeta que acabamos de crear mira el punto anterior `cd .`

cp (copy)

Para copiar ficheros (o carpetas) usamos el comando `cp` seguido del fichero (o carpeta) de origen, un espacio y la ruta del fichero (o carpeta) de destino:

```
cp fichero-de-origen.html carpeta-de-destino/fichero-de-destino.html
```

mv (move)

Para mover ficheros (o carpetas) usamos el comando `mv` seguido del fichero (o carpeta) de origen, un espacio y la ruta del fichero (o carpeta) de destino:

```
mv fichero-de-origen.html carpeta-de-destino/fichero-de-destino.html
```

Este comando también sirve para renombrar, ya que renombrar un fichero de `a.html` a `b.html` es lo mismo que moverlo.

```
mv a.html b.html
```

clear

A veces pasa que hemos introducido muchos comandos y sería genial poder limpiar la ventana. Para eso existe el comando `clear`.

```
clear
```

pwd (print working directory)

Principalmente usaremos la terminal para movernos por el sistema de archivos y carpetas del ordenador. Así que es fundamental saber dónde estamos en cada momento. El comando `pwd` se encargará de mostrarnos en qué carpeta nos encontramos. Si escribimos:

```
pwd
```

La terminal mostrará la ruta absoluta de la carpeta en la que estemos, con este aspecto:

```
/user/nombre-de-usuario
```

Nos estaría indicando que nos encontramos en la carpeta `nombre-de-usuario`, que está dentro de la carpeta `user`, que está en la carpeta raíz de nuestro equipo.

Si estás trabajando en un Ubuntu integrado dentro de Windows 10 y pruebas `pwd` verás que el resultado es:

```
/mnt/c/Users/nombre-de-usuario
```

Es decir, en Windows 10 las unidades de nuestro ordenador como `c:\` se montan dentro de `/mnt/`, por ello la ruta `c:\Users\maricarmen` corresponde con `/mnt/c/Users/maricarmen`.

Historial de comandos

Para movernos por los últimos comandos ejecutados usamos la teclas de flecha para arriba `↑` y para abajo `↓`. Así nos ahorraremos volver a escribir lo mismo muchas veces.

Ayuda y opciones

Si no sabemos cómo funciona un comando podemos **buscar en Internet (siempre muy útil)** o pediremos ayuda a la terminal. Por ejemplo para saber cómo funciona el comando `ls` escribimos la opción `--help`:

```
ls --help
```

El terminal mostrará una explicación de cómo se utiliza el comando y las opciones. Con esta información sabremos que para listar todos los ficheros de un directorio, incluidos los ocultos usaremos:

```
ls -a
```

Es decir, las letras que pongamos después del guion – son las opciones. Y podemos poner una o varias. Y todos los comandos tienen la opción de ayuda en --help .

A practicar

Durante los primeros días del curso te explicaremos en detalle cómo se usa la terminal y sus comandos. Pero es interesante que empieces a familiarizarte un poco, así que te recomendamos que hagas el siguiente ejercicio:

1. Crea una carpeta en tu ordenador como lo has hecho siempre:
 1. Botón derecho
 2. Crear carpeta
 3. ...
2. Abre la carpeta desde VS Code.
3. Abre una terminal y desde ahí practica algunos comandos como...
4. Crea una subcarpeta.
5. Entra en la subcarpeta.
6. Sube desde la subcarpeta a la carpeta superior.
7. Lista el contenido de la carpeta.
8. Trastea y prueba cosas...

Rutas de las carpetas y ficheros

Los proyectos de analítica de datos pueden estar compuestos de muchas carpetas y ficheros (imágenes, tablas de resultados, módulos con funciones, etc.) que están relacionados entre sí. **O dicho de otra forma unos ficheros usan otros ficheros.**

Un ejemplo sería la carpeta de imágenes, para introducirlas en un informe. Imaginemos que deseamos guardar un histograma. Para esto nosotras escribiríamos código de Python y al realizar el guardado de la imagen(también llamado exportar) lo realizaríamos dentro de su carpeta correspondiente. Dentro del código para guardar la imagen escribiríamos la ruta `./output/images/histogram.jpg`.

La definición de ruta es **el camino que hay que recorrer para ir desde una carpeta o fichero de origen a un carpeta o fichero de destino, para relacionarlos entre sí.**

También vamos a utilizar rutas cuando trabajemos con la terminal, para movernos de la carpeta en la que estamos trabajando en un momento dado a otra carpeta en la que queremos empezar a trabajar.

Hay rutas absolutas y rutas relativas.

Rutas absolutas

Las rutas absolutas indican la **dirección completa** de una carpeta o fichero **desde la raíz del ordenador**. Es decir el origen de la ruta es la carpeta principal del ordenador. El destino es el fichero que queremos enlazar:

- Siempre empiezan con `/`. La `/` es la ruta raíz del ordenador.
- Por ejemplo `/user/maricarmen/adalab/proyectos/modulo-1/`.
- Por ejemplo `/user/maricarmen/adalab/proyectos/modulo-1/python/`
- Por ejemplo `/mnt/c/Users/maricarmen/adalab/proyectos/modulo-1/python/leccion1.ipynb`

Rutas relativas

- Las rutas relativas indican **el camino que hay que recorrer** para ir desde la carpeta actual en la que estoy ahora mismo a otra carpeta o fichero. Cuando decimos "la carpeta actual en la que estoy" nos referimos a la carpeta en la que está nuestra terminal, o a la carpeta en la que está el fichero que estamos editando.
- La ruta relativa más simple es `.. /` (dos puntos barra). Esta ruta **indica la carpeta madre o carpeta superior respecto a la carpeta actual**. Si la ruta relativa es el camino a recorrer, escribiendo la ruta `.. /` vamos de la carpeta actual a la carpeta madre. Si escribimos la ruta `.. / .. /` vamos de la carpeta actual a la carpeta abuela. Y así sucesivamente...
- Otra ruta relativa muy simple es `. /` (punto barra). Esta ruta indica la carpeta en la que estoy, mi carpeta actual. Si la ruta relativa es el camino a recorrer, escribiendo la ruta `. /` vamos de la carpeta actual a la carpeta actual. Esto parece que no tiene mucho sentido, en seguida veremos que sí lo tiene.

Vamos a ver un ejemplo de rutas relativas. Teniendo esta estructura de carpetas y ficheros:

```
adalab/
├── proyectos/
│   └── modulo-1/
│       ├── python/
│       │   └── leccion1.ipynb
│       │   └── ejercicios-py-l1.ipynb
│       └── sql/
│           └── ejercicios-sql-l1.ipynb
└── modulo-2/
```

Pensemos que estoy en la carpeta `proyectos/` y quiero *caminar* hasta la carpeta `modulo-1/`. La ruta a usar es `./modulo-1/`. Es decir, desde `proyectos/` que es la actual, entro en `modulo-1/`.

Ahora pensemos que estoy en la carpeta `proyectos/` y quiero *caminar* hasta la carpeta `python/`. La ruta a usar es `./modulo-1/python/`. Es decir, desde `proyectos/` entro en `modulo-1/`. Después desde `modulo-1/` entro en `python/`.

Ahora pensemos que estoy en la carpeta `sql/` y quiero entrar en la carpeta `python/`. La ruta a usar es `../python/`. Primero subo a la carpeta madre que es `modulo-1/` pero no necesito indicar el nombre de la carpeta madre porque madre no hay más que una y me vale con poner `.. /`. Después desde la carpeta madre entro en la carpeta `python/`.

Rutas dentro de ficheros

Cuando nos movemos por la terminal podemos usar rutas absolutas o relativas. Pero **cuando escribimos la ruta desde dentro de un fichero a otro fichero debemos usar siempre rutas relativas**. Esto se debe a que cuando estamos programando una página, los ficheros están en mi ordenador. Pero si los comparto con una compañera, ella los colocará donde quiera, y su ruta absoluta será diferente a la mía.

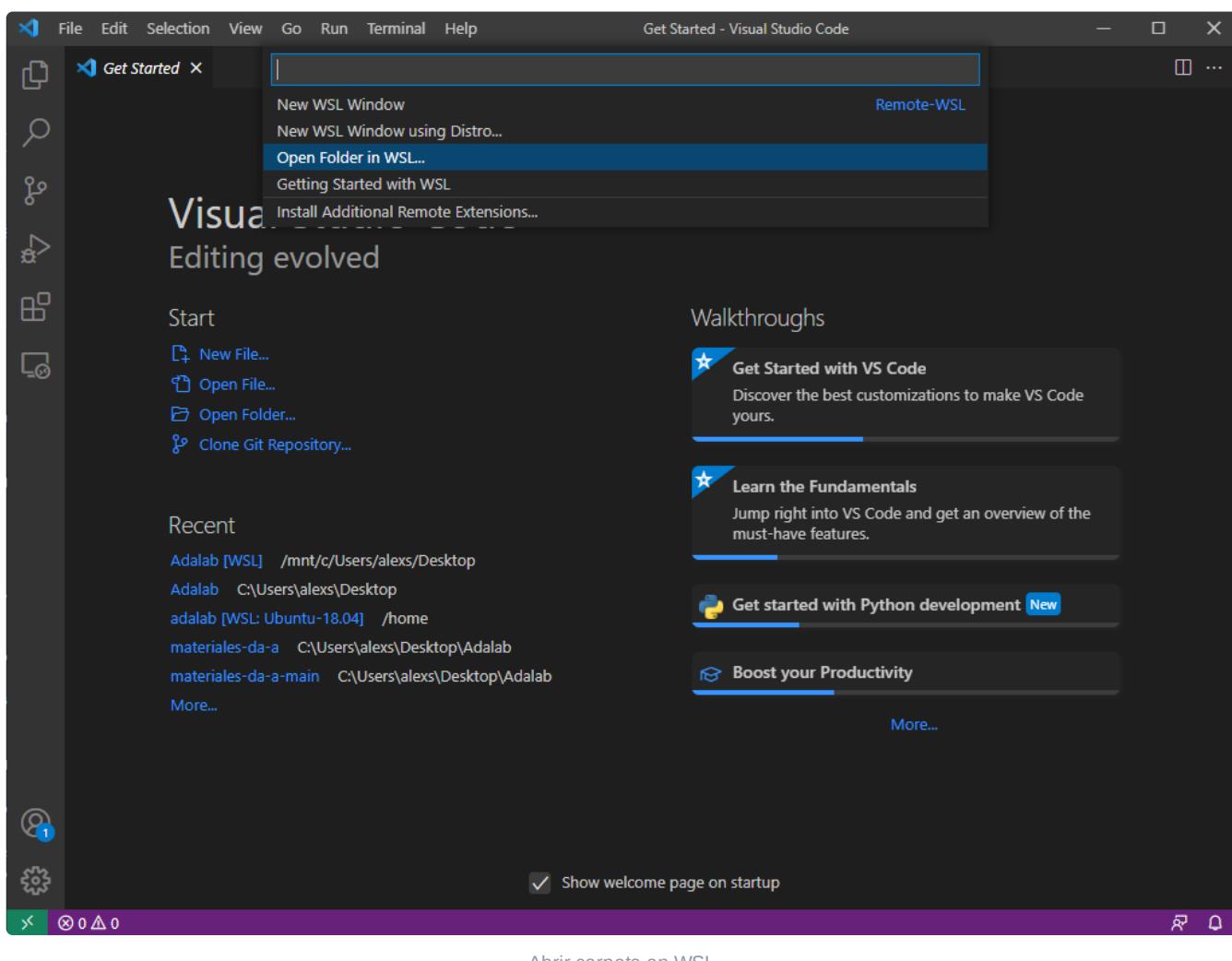
BRICONSEJO: Acuérdate de este importante consejo. Cuando pongas la ruta de un fichero dentro de otro, la ruta debe empezar con `./` o `.. /`. **Siempre. Sin excepciones.** Si alguna vez no lo haces, antes o después tendrás problemas y acabarás acordándote de este briconsejo.

Windows Subsystem for Linux - Anaconda

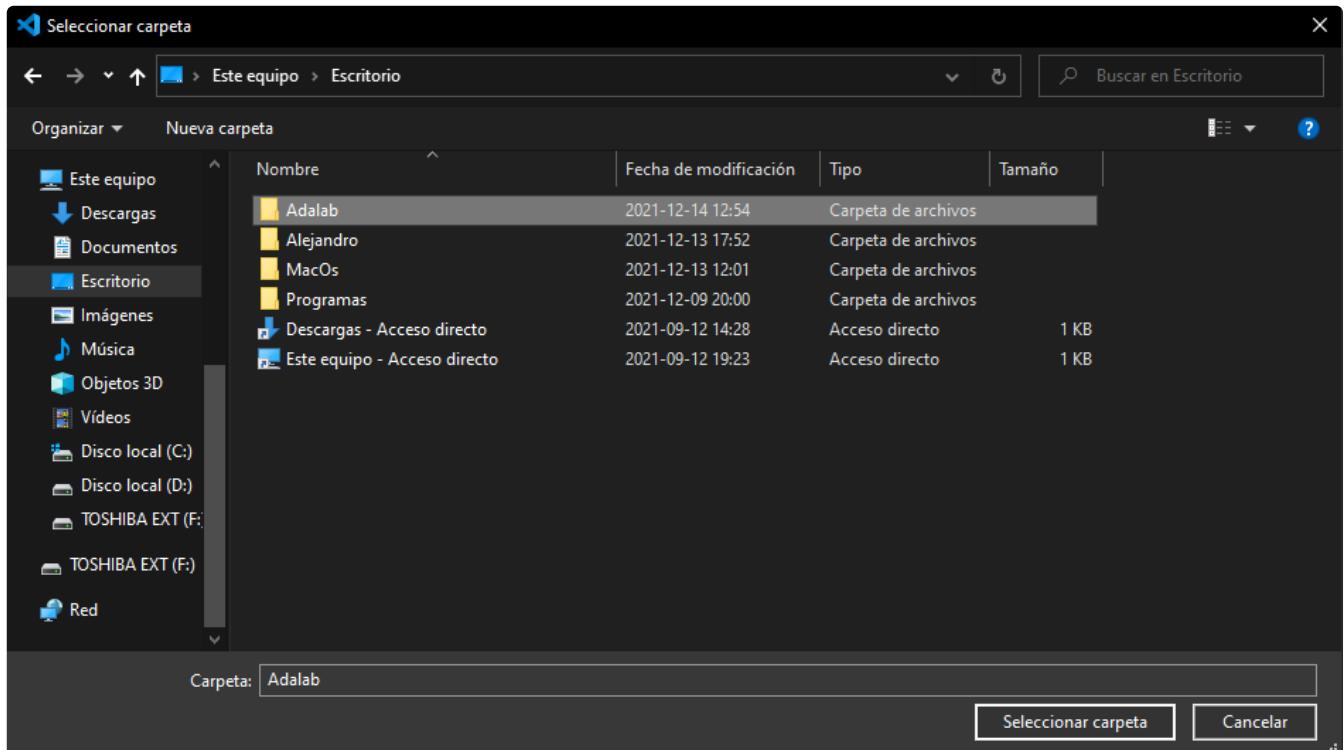
Este capítulo de la guía de instalación únicamente será necesario en caso de haber escogido el WSL (Windows Subsystem for Linux) para el uso de la terminal. En caso de estar utilizando Linux o Mac, puedes obviar este paso.

Habilitar WSL en VS Code para utilizar Anaconda y carpetas locales en Windows

1. Creamos en el escritorio(o en la ruta que deseas) una carpeta llamada Adalab, donde iremos guardando el contenido que utilizaremos a lo largo del curso.
2. Abrimos VS Code y clicamos en el icono  , tras esto aparecerá el siguiente desplegable en la ventana principal de VS Code y seleccionaremos **Open Folder in WSL**.

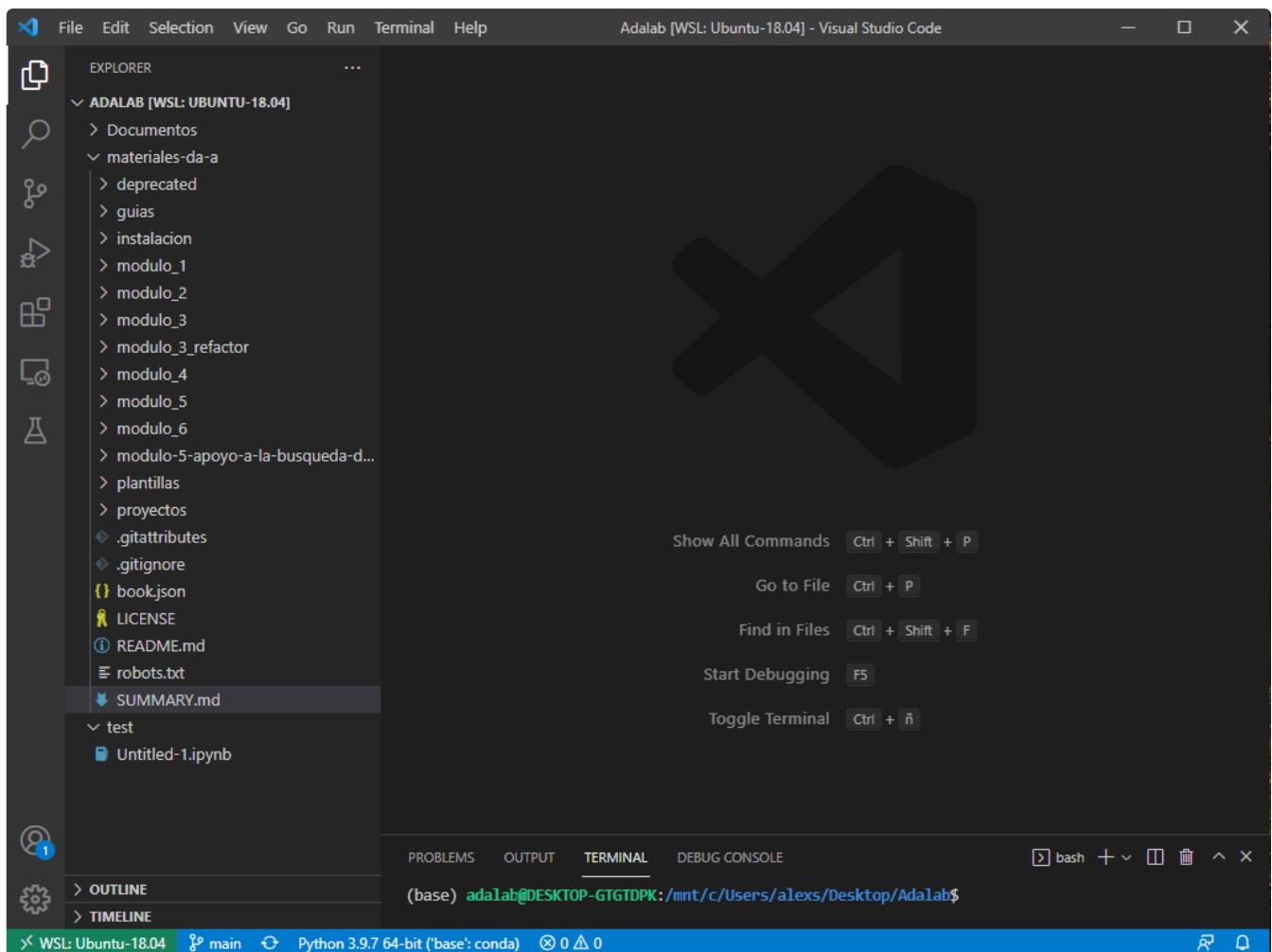


3. Vamos a la carpeta llamada Adalab y la seleccionamos.



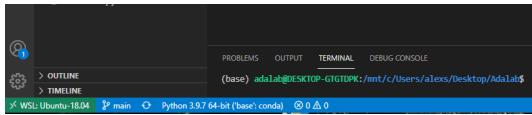
Selección carpeta Adalab

4. Nos debería quedar ahora la ventana de VS Code tal y como se muestra en la siguiente ventana:



Ventana VS Code con la carpeta cargada

Una vista con zoom de la parte inferior izquierda de la ventana de VS Code.



- ! Este proceso deberá ser repetido cada vez que deseemos cambiar la carpeta en la que estamos trabajando en el momento. Ya que si no, nos saltará un error de VS Code indicando que no tenemos Python instalado, ya que únicamente esta instalado en WSL.

Extensiones en WSL

Las [extensiones](#) de VS Code no siempre se instalan para el VS Code entero, sino a veces solo se instala local. Esto depende de la extensión, entonces revisa las extensiones que instalaste ya que tienes el Ubuntu facilitado: puede que te aparezca `Install in WSL: Ubuntu-18.04`. Para las en que sea así, instálalas de nuevo.

Entorno Anaconda Adalab

En este capítulo de la guía de instalación vamos a explicar como configurar en entorno de Anaconda para poder utilizar todas los paquetes necesarios durante el curso y evitar así tener que realizar instalaciones adicionales durante el desarrollo del bootcamp y evitar posibles inconvenientes futuros.

0. Descarga este archivo.

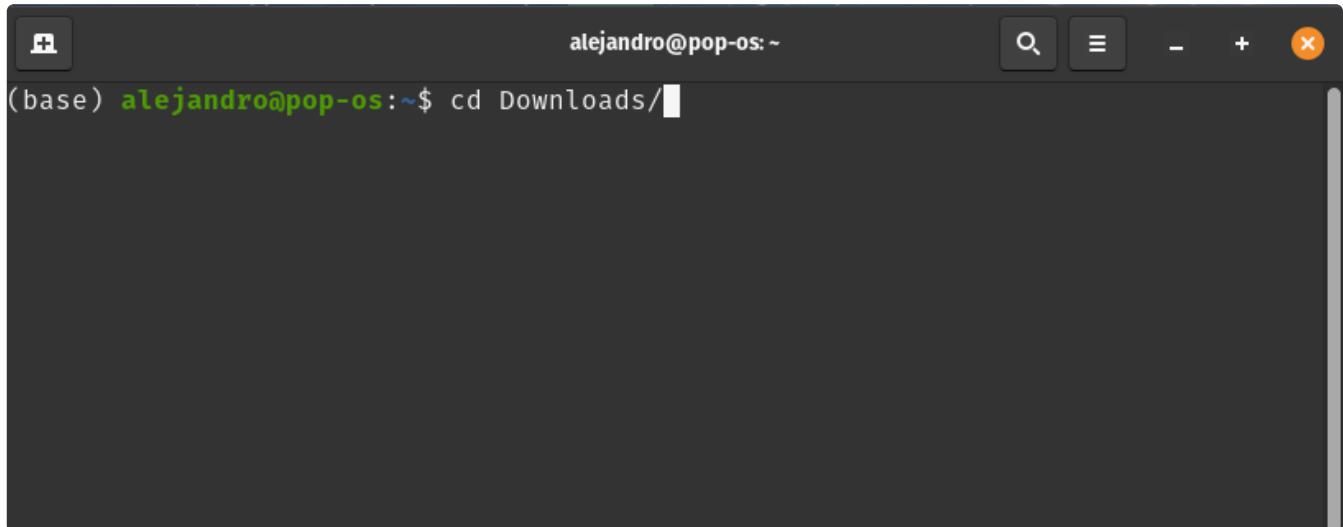
1. De esta forma lo primero que deberemos realizar es abrir una terminal.

- Si estamos en Ubuntu o Linux deberíamos poder abrirla directamente.
- Si estamos en windows con el WSL deberemos abrir el Vscode y abrir una terminal.

Y escribir lo siguiente:

```
cd Downloads/
```

Nos debería salir algo así al abrir la terminal y escribir el texto anterior:



A screenshot of a terminal window titled 'alejandro@pop-os: ~'. The command '(base) alejandro@pop-os: ~\$ cd Downloads/' is visible in the terminal. The window has a dark theme with light-colored text and icons.

Abrir terminal

2. Escribimos ahora:

```
cd conda create --name adalabenv python=3.10
```

Y le damos a enter. Debería quedarnos algo similar a lo mostrado en la imagen siguiente:



Tras esto nos pedirá si deseamos continuar y le diremos que si. Para ello escribiremos `y` (si tenemos el equipo en idioma inglés) o alternativamente `s` (si tenemos el equipo en idioma Español)

```
alejandro@pop-os: ~/Downloads
bzip2          pkgs/main/linux-64::bzip2-1.0.8-h7b6447c_0
ca-certificates pkgs/main/linux-64::ca-certificates-2022.07.19-h06a4308_0
certifi         pkgs/main/linux-64::certifi-2022.9.14-py310h06a4308_0
ld_impl_linux-64 pkgs/main/linux-64::ld_impl_linux-64-2.38-h1181459_1
libffi          pkgs/main/linux-64::libffi-3.3-he6710b0_2
libgcc-ng       pkgs/main/linux-64::libgcc-ng-11.2.0-h1234567_1
libgomp         pkgs/main/linux-64::libgomp-11.2.0-h1234567_1
libstdcxx-ng   pkgs/main/linux-64::libstdcxx-ng-11.2.0-h1234567_1
libuuid         pkgs/main/linux-64::libuuid-1.0.3-h7f8727e_2
ncurses         pkgs/main/linux-64::ncurses-6.3-h5eee18b_3
openssl         pkgs/main/linux-64::openssl-1.1.1q-h7f8727e_0
pip             pkgs/main/linux-64::pip-22.1.2-py310h06a4308_0
python          pkgs/main/linux-64::python-3.10.4-h12debd9_0
readline        pkgs/main/linux-64::readline-8.1.2-h7f8727e_1
setuptools      pkgs/main/linux-64::setuptools-63.4.1-py310h06a4308_0
sqlite          pkgs/main/linux-64::sqlite-3.39.2-h5082296_0
tk               pkgs/main/linux-64::tk-8.6.12-h1ccaba5_0
tzdata          pkgs/main/noarch::tzdata-2022c-h04d1e81_0
wheel           pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
xz              pkgs/main/linux-64::xz-5.2.5-h7f8727e_1
zlib            pkgs/main/linux-64::zlib-1.2.12-h5eee18b_3

Proceed ([y]/n)?
```

instalacion entorno

Como se muestra en la siguiente imagen tras unos segundos se habrá instalado nuestro entorno.

3. Procedemos a activar el entorno escribiendo el siguiente código y le damos a enter.

```
conda activate adalabenv
```

Podemos ver el resultado en la siguiente imagen.

```
alejandro@pop-os: ~/Downloads
tzdata          pkgs/main/noarch::tzdata-2022c-h04d1e81_0
wheel           pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
xz              pkgs/main/linux-64::xz-5.2.5-h7f8727e_1
zlib            pkgs/main/linux-64::zlib-1.2.12-h5eee18b_3

Proceed ([y]/n)? y

Downloading and Extracting Packages
certifi-2022.9.14 | 155 KB    | #####| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate adalabenv
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) alejandro@pop-os:~/Downloads$ conda activate adalabenv
```

Cambio a adalabenv

4. Finalmente una vez estemos en el entorno adalabenv, escribimos lo siguiente

```
pip install -r requirements.txt
```

Y de damos de nueva a la tecla enter. Nos debería de salir algo semejante a lo mostrado en la imagen y dejamos que termine el proceso.

```
(adalabenv) alejandro@pop-os:~/Downloads$ pip install -r requirements.txt
Collecting alabaster==0.7.12
  Using cached alabaster-0.7.12-py2.py3-none-any.whl (14 kB)
Collecting anaconda==0.0.1.1
  Using cached anaconda-0.0.1.1-py3-none-any.whl
Collecting anaconda-client==1.2.2
  Using cached anaconda_client-1.2.2-py3-none-any.whl
```

Abrir carpeta en WSL

Una vez termine, ya tendremos configurado el entorno que utilizaremos en clase, ya tenemos todo configurado por ahora, perfecto!

Curso introductorio al mundo data y la terminal

Esta sección es una de las importantes de la semana previa de trabajo. Os facilitamos un para que podáis comprender los diferentes roles que existen en el mundo data y otro para empezar a enfrentarse a terminal y la línea de comandos. **Estudiar y comprender esta información básica os ayudará a entender desde el día 0 vuestro rol como futuras analistas de datos.**

Entendemos que esto puede ser un desafío, ¡pero no te preocupes!, todos los conocimientos que contienen estos cursos los vamos a explicar en profundidad en el curso.

Como alumna del bootcamp de data analytics tendrás acceso gratuito a la plataforma de [Platzi](#) donde encontrarás los siguientes cursos:

- [Curso Introductorio mundo data](#)
- [Introducción a la Terminal y Línea de Comandos](#)
- [Curso de Entorno de Trabajo para Ciencia de Datos con Jupyter Notebooks y Anaconda](#)

Una vez rellenes el [formulario para acceder a Platzi](#), pueden tardar algunos días en darte de alta. Si al cabo de una semana aún no tienes acceso a la plataforma y tampoco encuentras el correo en la carpeta de Spam. Por favor, ponte en contacto con el equipo de Platzi indicando que eres Adalaber.

Control de versiones Git

Introducción de Git/GitHub

Nota: Es imprescindible que realices esta sección en la semana previa de trabajo.

Introducción a Git y GitHub

Con el objetivo de empezar a trabajar de manera óptima en remoto desde el primer día de curso vamos a aprender lo básico sobre Git y GitHub. En los primeros días de clase también os lo explicaremos en detalle.

Qué es Git



Git

Git es un sistema de **control de versiones** de código además de una **herramienta para compartir** código.

El control de versiones nos sirve para ver el histórico de cambios que hemos hecho en el código. Es útil pasar saber por ejemplo que hace un mes nuestra compañera Mari Carmen añadió un determinado código al proyecto. **Git nos dice quién, cuándo, por qué y dónde se cambió cada línea de código de un proyecto.**

También es una herramienta para compartir código entre nuestras compañeras de programación de los proyectos de Adalab, nuestros profes o la empresa en la que trabajaremos. **Es como el Drive o Dropbox de los programadores.**

Qué es GitHub



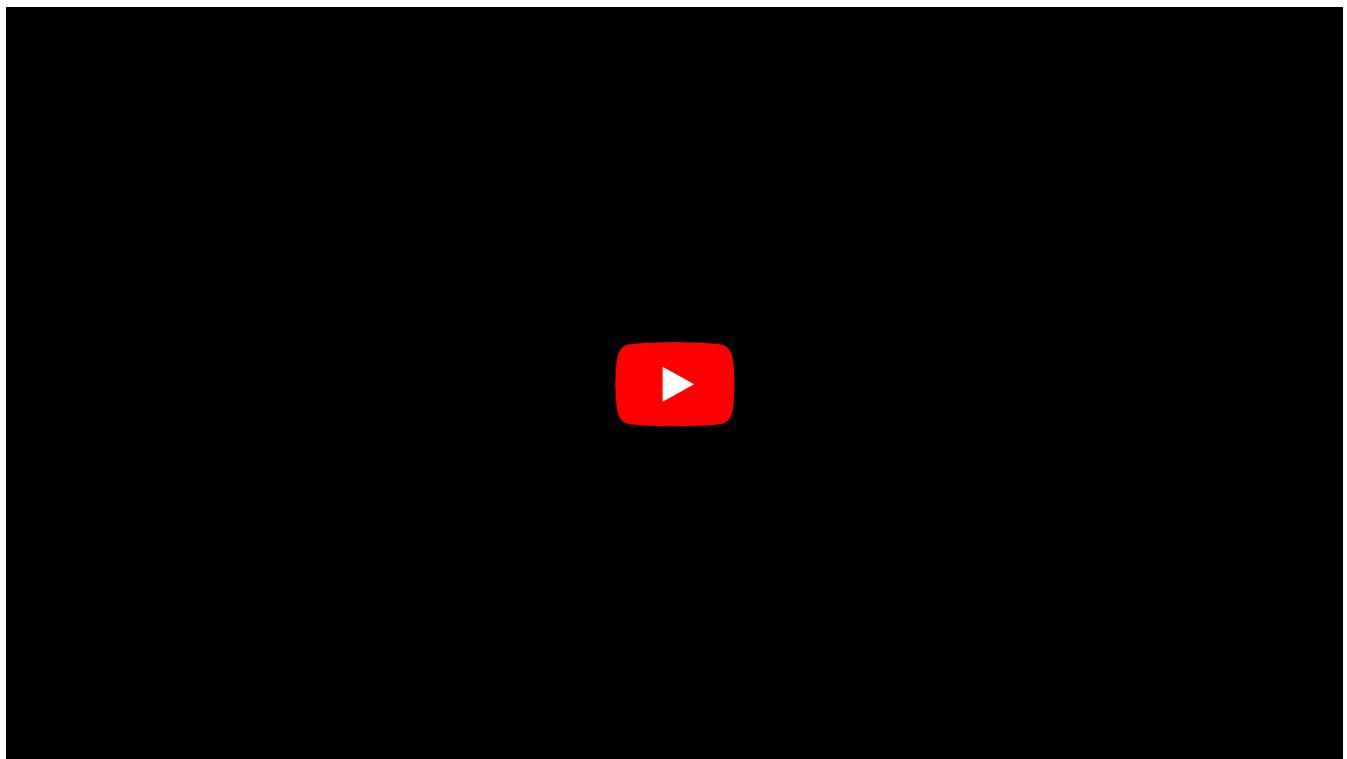
GitHub

Si Git es el sistema o programa para compartir código, [GitHub](#) es una de tantas empresas que hay para usar Git. A través de su web podemos crear proyectos y compartirlos con otras personas. También funciona como red social de programación.

Muchos de los servicios que ofrece GitHub (que por cierto ha sido recientemente comprada por MicroSoft) son gratuitos y otros son de pago.

Cómo clonar un repositorio de otra persona

Vamos a empezar por clonar un repositorio de otra persona. En concreto vamos a clonar el repositorio donde las profes subimos los ejercicios hechos en clase. De esta forma, al acabar cada clase podrás descargarte el código de dichos ejercicios:



Los comandos utilizados en [este vídeo](#) son:

```
git clone https://github.com/adalab/ejercicios-en-clase-X
```

```
git pull
```

Crea un token en GitHub

Hasta hace pocas semanas para subir tu código a GitHub solo era necesario tu nombre de usuaria y tu contraseña. Actualmente GitHub está cambiando los procesos para subir el código y dependiendo de la versión de Git que tengas instalada en tu ordenador, el sistema operativo y otras cosas, GitHub te pedirá tu contraseña o un token (que es una contraseña generada por ellos). Por este motivo lo primero que vamos a hacer es **crear un token**:

- Entra en tu GitHub.
- Despliega en tu menú (esquina superior derecha) y pulsa en **Settings**.
- Pulsa en **Developer settings** (en el menú vertical de la izquierda).
- Pulsa en **Personal access tokens** (en el menú vertical de la izquierda).
- Pulsa en **Generate new token**.
- En **Note** escribe una descripción para tu token, por ejemplo: **Autenticación desde mi ordenador personal**. Esta nota es para que en el futuro recuerdes para qué es este token.
- Marca la primera opción **repo**.

[Settings](#) / Developer settings

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Autenticación desde mi ordenador personal

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry

GitHub token

- Guarda tu token pulsando en **Generate token**.
- GitHub te mostrará tu token (un código largo y raro). Guárdalo en lugar seguro porque lo necesitarás en el futuro.

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

X

Settings / Developer settings

GitHub Apps

OAuth Apps

Personal access tokens

Personal access tokens

Personal access tokens

Generate new token

Revoke all

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

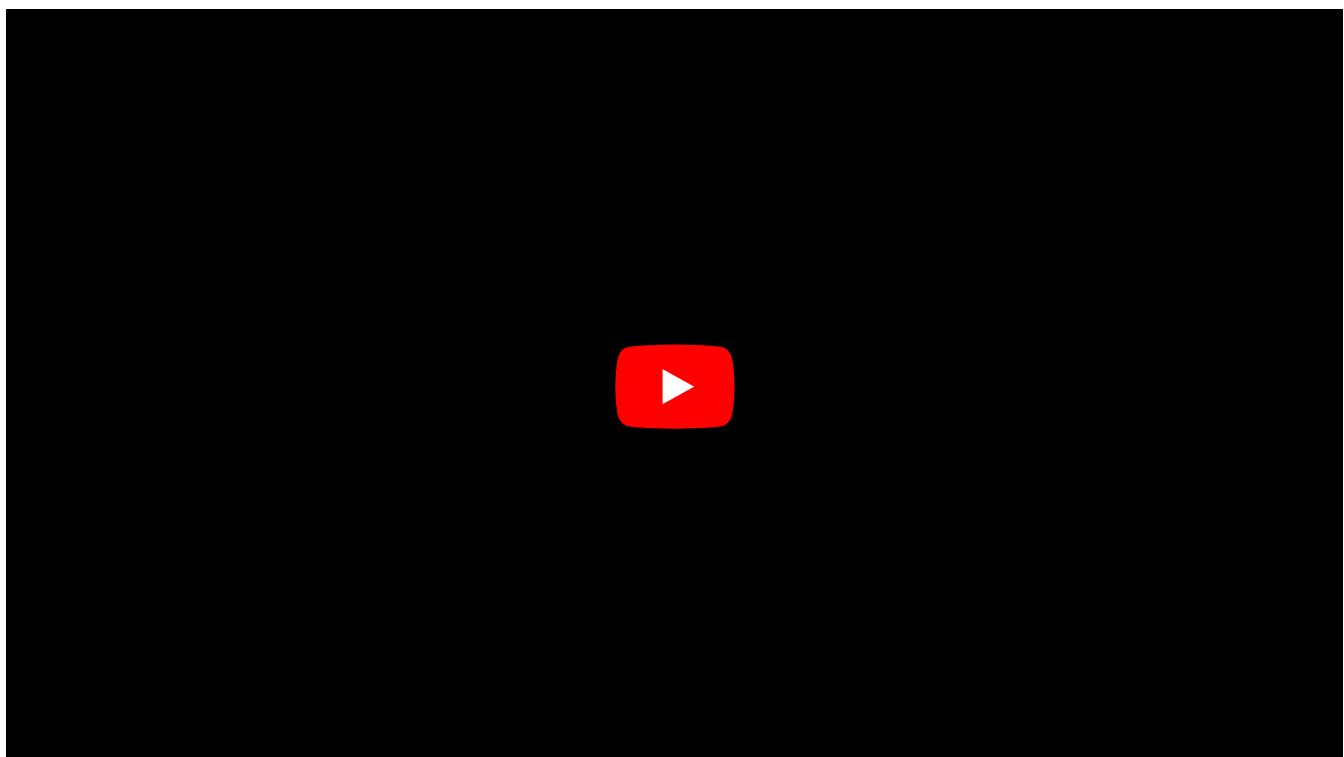
✓ ghp_EetqpyXF0L0c17bbUmFdEki1weZk1L233cSg ⌂

Delete

GitHub token

Cómo crear mi repositorio de Git

A continuación te pedimos que crees tu propio repositorio en GitHub donde durante el curso irás subiendo los ejercicios que hagas en clase. Al subirlos a la nube podrás compartirlos con tus compañeras y ellas contigo. En programación lo compartimos todo!!!



Los comandos utilizados en [este vídeo](#) son:

```
git add -A
```

```
git commit -m "Mensaje del commit"
```

```
git push
```

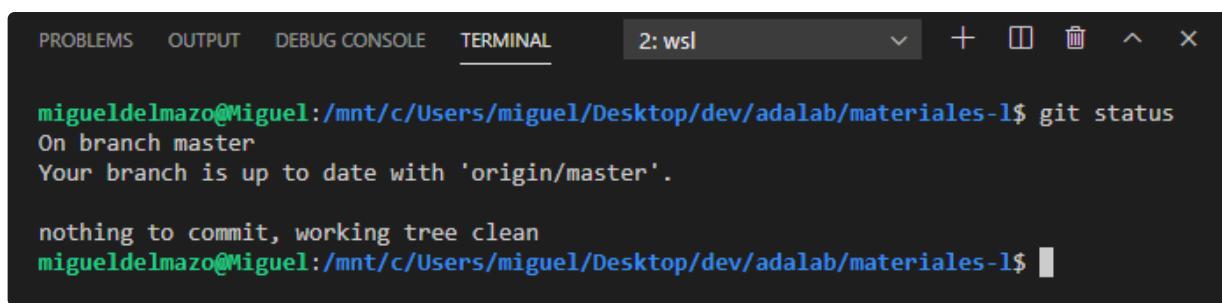
Al hacer `git push` por primera vez en tu terminal, ésta te pedirá que metas tu usuario de GitHub y tu contraseña. En función de lo que GitHub decida tienes que meter tu contraseña de GitHub o el token que has generado en el apartado anterior. **Prueba a meter tu contraseña y si funciona pues guay. Si no te funciona, repite el proceso metiendo tu token.**

Git status

Ya sabemos usar los comandos de Git `clone`, `add`, `commit`, `pull` y `push`. Solo nos queda por aprender un comando importante: `git status`.

`git status` nos dice el estado actual en el que está nuestro repo. Es muy útil para saber lo que tenemos que hacer, o porqué estamos teniendo un problema que no nos deja avanzar.

Supongamos que yo acabo de subir mi código con un `git add -A`, `git commit -m "My message"` y `git push` al repositorio remoto. Es decir mi repo local y el remoto están igual. Si yo escribo el comando `git status` me aparecerá:



A screenshot of a terminal window titled '2: wsl'. The window shows the command 'git status' being run in a Linux environment. The output indicates that the user is on branch 'master', the branch is up-to-date with 'origin/master', and there is nothing to commit because the working tree is clean.

```
migueldelmazo@miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
migueldelmazo@miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$
```

Git status

Es decir, `git status` me está diciendo que estoy actualizado.

A continuación decidí modificar el fichero de mi repo `guias/intro_a_git_github.md` y añadido un fichero nuevo que se llama `guias/assets/images/git-status-with-changes.png` y ejecuté el comando `git status`. En la terminal aparecerá algo como:

```
miguelelmazo@Miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   guias/intro_a_git_github.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    guias/assets/images/git-status-up-to-date.png

no changes added to commit (use "git add" and/or "git commit -a")
miguelelmazo@Miguel:/mnt/c/Users/miguel/Desktop/dev/adalab/materiales-1$
```

Git status

Es decir, me está diciendo que tengo un fichero modificado y un fichero que acabo de añadir. Además me sugiere varios comandos que puedo utilizar para añadir los cambios al repo o para deshacer los cambios.

Es importante que para acostumbrarte a utilizar Git y para asimilar bien cómo funciona internamente, cada vez que ejecutes un comando leas lo que responde la terminal. Y después de ejecutar un comando cualquiera ejecuta `git status` para ver en qué estado estás.

Repositorios de Adalab

Durante el curso vamos a utilizar los siguientes repositorios de Git / GitHub. Te recomendamos que los clones en tu ordenador y hagas `git pull` frecuentemente.

- <https://github.com/adalab/ejercicios-en-clase-q>: repositorio donde las profesoras vamos a subir los ejercicios hechos en clase de esta promoción.
- <https://github.com/adalab/soluciones-alumnas-q>: repositorio para que alumnas subáis las soluciones de los ejercicios que queráis compartir con las compañeras. Os recomendamos que cuando terminéis un ejercicio lo subáis a este repositorio, las compañeras os lo agradecerán.
- <https://github.com/adalab/ejercicios-extra>: repositorio con enunciados y ejercicios extras resueltos, de todas las promociones.
- <https://github.com/adalab/adalab-web-starter-kit>: repositorio con el proyecto base que utilizaremos en el módulo 1 y 2.
- <https://github.com/adalab/recursos-de-las-alumnas>: recursos técnicos creados por las alumnas.

Estructura de carpetas

En Adalab vamos a crear muchas carpetas para proyectos, ejercicios, evaluaciones... Te recomendamos que crees una carpeta en tu ordenador que se llame **Adalab** con las siguientes subcarpetas:

- `ejercicios-de-adalab`:
 - En los vídeos anteriores has creado esta carpeta al clonar tu primer repositorio. Mueve la carpeta entera aquí.
 - Dentro de esta carpeta crea una subcarpeta por cada ejercicio que vayas haciendo durante el curso. Te recomendamos que las carpetas de los ejercicios se llamen `modulo-x-leccion-y-ejercicio-z-descripcion`. Por ejemplo:
 - `modulo-1-leccion-04-ejercicio-03-flexbox`
 - `modulo-1-leccion-10-ejercicio-10-grid`
 - Si todas las alumnas seguís este patrón es muy fácil compartir ejercicios y códigos entre vosotras.
 - Por cierto, cada ejercicio que hagamos lo metemos en una carpeta diferente. **Nunca se deben hacer dos ejercicios en la misma carpeta**, da muchos errores. **Insistimos, nunca haremos dos ejercicios en la misma carpeta**.
- `proyectos` : aquí meterás los proyectos que haremos durante el curso.
- `evaluaciones` : aquí meterás las evaluaciones que haremos durante el curso.
- `ejercicios-en-clase-q` : clona este [repositorio](#) aquí.
- `soluciones-alumnas-q` : clona este [repositorio](#) aquí.
- `ejercicios-extra` : clona este [repositorio](#) aquí.
- `adalab-web-starter-kit` : clona este [repositorio](#) aquí.
- `recursos-de-las-alumnas` : clona este [repositorio](#) aquí.
- Otros: por supuesto puedes añadir otras carpetas que quieras, por ejemplo con los apuntes que vayas tomando durante del curso...

Y para terminar te damos dos consejos:

- No metas un repositorio de Git dentro de otro repositorio de Git. Da muchos problemas. Por ello siempre que clones un repo nuevo hazlo en las carpetas **Adalab, Proyectos o Evaluaciones**.
- A mí mi abuela (una mujer muy sabia) siempre me decía que **es de buena programadora ser ordenada**. Mantén tus carpetas, códigos y demás bien organizados. Tu cerebro te lo agradecerá.

Avanzado de Git/GitHub

Nota: no es necesario que leas esta guía antes de empezar el curso de Adalab, te lo iremos enseñando poco a poco. La idea es que vengas aquí para resolver problemas puntuales.

En esta guía **estamos recopilando todo lo que enseñamos en el curso** sobre Git y GitHub. Mucha de esta información y de estos vídeos están también en algunas de las lecciones de los módulos 1 y 2.

Git es una herramienta que **se aprende con el uso y la práctica**.

Esta guía está dividida en dos partes: principales comandos de Git y vídeos.

Principales comandos de Git

git command --help

Obtener ayuda sobre un comando:

```
git clone --help
```

```
git commit --help
```

git clone

Clonar un repo en nuestro ordenador desde un repo remoto:

```
git clone url-de-repo-en-github
```

git add

Añadir todos los ficheros modificados al próximo commit:

```
git add -A
```

Añadir algunos ficheros modificados al próximo commit:

```
git add nombre-de-fichero
```

```
git add nombre-de-una-carpeta/*
```

git commit

```
git commit -m "Add footer styles"
```

git push

Subir uno o varios commits al repositorio remoto:

```
git push
```

Subir los últimos commits al remoto después de crear una rama:

```
git push -u origin nombre-de-rama
```

git pull

Bajar los últimos commits desde el repositorio remoto:

```
git pull
```

git status

Ver el estado de nuestro repo local:

```
git status
```

git fetch

Sincroniza la información (de commits, ramas...) desde el repo remoto al local, sin hacer ningún cambio en nuestro código ni en nuestro historial local de commits, lo podemos hacer siempre que queramos:

```
git fetch
```

git clean

A veces, creamos ficheros que no hacen falta y no van a ir a ningún commit. Si queremos limpiar nuestro repositorio de esos ficheros podemos usar `git clean`:

```
git clean -f .
```

También podemos usar `git clean -i .` para que nos vaya preguntando cuál fichero borrar y cuál no.

Si esos ficheros extra no queremos que vayan en ningún commit pero no los vamos a borrar porque son necesarios (por ejemplo, ficheros de configuración o el directorio de configuración de VisualStudio Code) podemos incluir sus nombres (con su ruta relativa) en el fichero `.gitignore` uno por línea. Si `.gitignore` no existe, podemos crearlo. Después, hacemos add y commit de `.gitignore` y a partir de entonces git no los tendrá en cuenta.

git reset

Si lo que queremos es deshacer todos los cambios que hemos hecho desde el último commit y quedarnos con el directorio "limpio de cambios", ejecutaremos:

```
git reset --hard
```

Usando `git reset` en conjunto al comando anterior `git clean`, que nos quedará el directorio de trabajo igualito a lo que había hasta el último commit.

Si queremos volver a commits previos, borrando los cambios realizados desde ese commit, podemos utilizar el siguiente comando:

```
git reset --hard [commit ID]: regresa hasta el commit de ID
```

git branch

Listar las ramas locales:

```
git branch
```

Listar las ramas remotas:

```
git branch -r
```

Crear una rama (sin movernos a ella):

```
git branch nombre-de-la-nueva-rama
```

git checkout

Movernos desde la rama actual a otra rama:

```
git checkout nombre-de-otra-rama
```

Crear y movernos a una rama en un solo comando:

```
git checkout -b nombre-de-la-nueva-rama
```

Tambien el comando git checkout + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero.

```
git checkout [commit ID] -- path/to/file
```

Si queremos mantener los cambios, una vez que estamos el archivo, debemos confirmar ese cambio. Esto se hace con el comando de confirmación estándar:

```
git commit -m 'confirmar mensaje'
```

git merge

Mover el código desde la rama de origen hacia la rama en la que estoy:

```
git merge nombre-de-rama-origen
```

git log

Ver el listado de commits realizados:

```
git log
```

Para salir de `git log` y volver a la terminal hay que pulsar `q`. Para ver más resultados, pulsa el espacio.

git diff

Sirve para ver los ficheros modificados y sus diferencias desde el último commit:

```
git diff
```

Aunque lo podemos usar para ver las diferencias entre dos commits:

```
git diff hash-commit-más-antiguo hash-commit-más-nuevo
```

...donde `hash-commit-más-antiguo` y `hash-commit-más-nuevo` son los identificadores de commit. Se ven después de hacer `git commit` o al listarlos con `git log`. Los puedes copiar y pegar o escribir sólo las primeras 5 letras.

También se puede usar para ver los cambios entre dos ramas:

```
git diff rama-1 rama-2
```

...nos dirá los cambios de la rama-1 que no están en la rama-2 con un símbolo - al principio de la línea y los de la rama-2 que no están en la rama-1 con un símbolo +.

Hay un nombre de commit "especial" que es **HEAD**. Automáticamente, git asigna al nombre HEAD el último commit de la rama en la que estamos. Es útil para ver los cambios que hemos hecho hasta ahora:

```
git diff hash-de-commit-antiguo HEAD
```

git commit --amend

Imaginaos que, por poco probable que parezca, nos hemos equivocado en el commit anterior: la descripción no es correcta o falta algún fichero. Podemos corregirlo modificando (enmendando) el último commit:

```
git commit --amend -m "Otra descripción"
```

```
git add fichero-que-me-deje.js  
git commit --amend
```

En el primer caso cambiamos la descripción del último commit y en el segundo añadimos un fichero más.

git revert

Si el último commit no era correcto porque tenía bugs o fallos y queremos desestimarlos, podemos usar git revert:

```
git revert
```

Lo que hace es generar otro commit nuevo con los cambios inversos al último commit y con la descripción del último commit pero con la palabra Revert al principio y los archivos como estaban antes de ese commit:

```
ivan@adalab:~/ejemplo-git$ git log --oneline
6c14dda (HEAD -> master) Cambio chungo
43da3d1 Otro cambio
c725fe8 Un cambio

ivan@adalab:~/ejemplo-git$ git revert HEAD
[master 466e605] Revert "Cambio chungo"
 1 file changed, 1 deletion(-)

ivan@adalab:~/ejemplo-git$ git log --oneline
466e605 (HEAD -> master) Revert "Cambio chungo"
6c14dda Cambio chungo
43da3d1 Otro cambio
c725fe8 Un cambio
```

git init

Crea un repositorio desde cero en nuestro ordenador, sin necesidad de crearlo previamente en GitHub:

```
git init
```

git add remote

Enlaza un repositorio de nuestro ordenador con un repositorio remoto:

```
git remote add origin url-del-repositorio-que-me-da-github
```

Vídeos de Git

Crear un repositorio

[Vídeo](#) sobre cómo crear y clonar un repositorio:



Añadir cambios a un repositorio

[Vídeo](#) sobre cómo añadir commits a un repositorio con los comandos `add`, `commit`, `push` y `pull`:



Merges blandos y duros

[Vídeo](#) sobre cómo trabajar dos personas en paralelo en un repositorio y cómo solucionar merges blandos (cuando ambas personas tocan diferentes líneas de código) y merges duros (cuando ambas personas tocan la misma línea):



Trabajar con varias ramas

[Vídeo](#) sobre cómo trabajar con varias ramas, movernos entre ramas, publicarlas y mergear una rama en otra:



Flujo de ramas

[Vídeo](#) sobre qué ramas debemos usar y cómo y cuándo debemos mover el código entre las diferentes ramas para mantener la mejor calidad de código:



Guías

Guía trabajo en remoto

Introducción al trabajo en remoto

Gran parte del tiempo que las alumnas dedicáis al curso lo hacéis en remoto. Por ello una de las soft skills que aprendemos en Adalab es a **trabajar bien en remoto**.

Saber trabajar en remoto tiene muchas ventajas, ya que nos permite trabajar desde diferentes sitios, pero además nos permite trabajar para empresas de cualquier parte del mundo. Y para trabajar bien lo más importante son 3 cosas:

- Tener una buena comunicación.
- Cumplir con los acuerdos y procesos a los que haya llegado el equipo de trabajo.
- Conocer el potencial de las herramientas que usamos en el día a día.

Los primeros días de clase **os daremos una charla sobre cómo trabajar en remoto** de forma óptima en una empresa.

Vamos a explicar cuáles son los procesos que usamos en Adalab.

Canales de comunicación

Para trabajar en remoto en Adalab debes saber lo siguiente:

Comunicación por chat

Toda la comunicación por chat, ya sea con las profesoras, personal de Adalab y resto de compañeras **la haremos por Slack**. No usamos emails ni otros chats como el de Zoom. Si usas otros chats seguramente el receptor no lea tu mensaje. Si en alguna ocasión necesitamos que uses otra vía de comunicación te avisaremos.

Es muy importante sacarle el máximo partido a herramientas como Slack, por ello durante los primeros días de clase te explicaremos cómo conocer todos sus trucos.

Comunicación por vídeo

Toda comunicación por vídeo la haremos por Slack y Zoom. Zoom lo usamos para las clases y comunicaciones colectivas y Slack para cuando estamos hablando un grupo pequeño de personas.

Es importante que cuando estemos hablando por vídeo **siempre activemos la webcam** ya que el poder vernos las caras mejora mucho la comunicación, somos capaces de percibir y expresar más información y podemos entender y empatizar más con el resto de personas.

Utilizamos Zoom porque nos permite grabar las clases en vídeo y así podéis volver a verlas posteriormente.

También hay que tener cuidado con los micrófonos. Si estamos en una reunión con mucha gente hay que procurar silenciar el micrófono si no estamos hablando, para evitar ruidos de fondo y acoplamientos de sonido.

Trabajo en remoto durante las horas de clase

A continuación vamos a explicar cómo nos vamos a conectar a través de Zoom. Para ello debemos conocer dos conceptos:

- **Salas de Zoom:** son salas normales de videollamadas iguales que las de otros servicios como pueden ser Slack, Skype, Hangout...
 - Disponen de las típicas funcionalidades como compartir pantalla, pintar en la pantalla de la persona que está compartiendo, permitir el control remoto...
- **Breakout rooms de Zoom:** Zoom dispone de una funcionalidad que permite hacer mini salas o subsalas dentro de una sala.
 - Sirven para que las profesoras agrupemos a las alumnas por equipos o por parejas.
 - Las profesoras podemos entrar y salir de estas subsalas para por ejemplo dar soporte a un equipo.
 - El concepto es similar al de una aula de verdad, en la que la profesora puede estar explicando para toda la clase una lección y luego pedir a las alumnas que trabajen por equipos. Si un equipo necesita ayuda, la profesora se acerca a la mesa del equipo a dar soporte.

En Adalab contamos con las siguientes salas de Zoom:

- Promo B: sala principal
- Promo B: sala de pair programming
- Promo B: sala de Alejandro
- Promo B: sala de Ana

Vamos a utilizar la sala **Promo A: sala principal** cuando estemos dando clase para todas las alumnas. En esta sala también estaremos las tres profesoras. También utilizaremos esta sala para las sesiones de desarrollo profesional.

Vamos a utilizar las **Promo A: sala de pair programming** para cuando estemos trabajando en parejas.

Lo más importante que debemos saber es que al empezar el día a las 8:30 de la mañana, después de cada descanso y los días que haya clases de desarrollo profesional, **siempre, siempre, siempre** debemos entrar en la **Promo A: sala principal**. En ese momento las profesoras os diremos qué vamos a hacer a continuación y os diremos a qué sala debe ir cada persona.

A tener en cuenta

- Para que te podamos enviar a las mini salas es importante que **el email de Gmail con el que te has registrado en Zoom sea el mismo que nos has facilitado a Adalab.**
- Si por lo que sea te sales de una mini sala y no puedes volver a entrar, lo que debes hacer es salir de la reunión de Zoom y volver a entrar. Aparecerás en la sala principal y la profesora te meterá en la mini sala que te corresponda.

Trabajo en remoto fuera de las horas de clase

Para trabajar en remoto fuera del horario de clase os recomendamos:

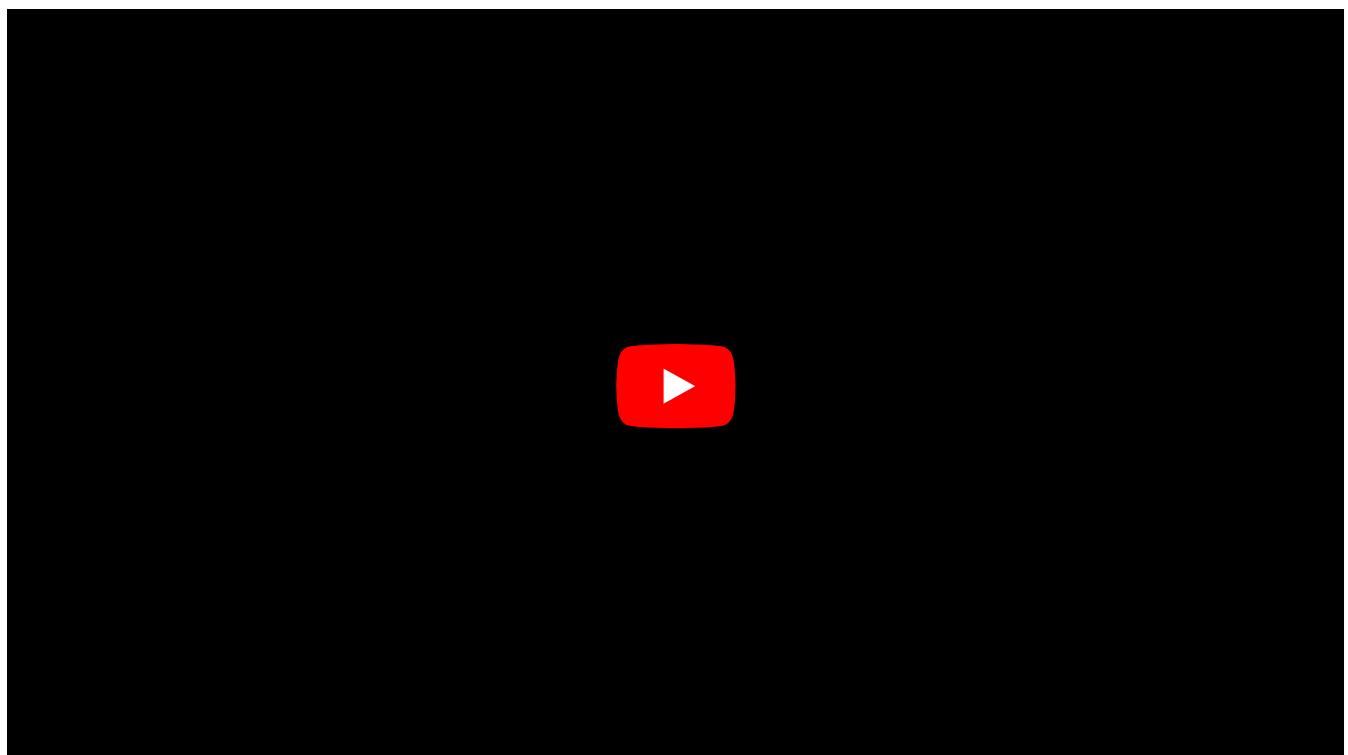
- Para comunicarte por chat usa Slack.
- Para comunicarte por vídeo usa:
 - **Slack para grupos pequeños**, por ejemplo con tu pareja o equipo.
 - La sala de Zoom **Promo A: sala principal** para grupos grandes. Las alumnas solo debéis usar esta sala cuando no se esté usando para dar clase.

Bonus: herramientas

Live Share de VS Code

Una herramienta estupenda para programar en remoto es [Live Share de VS Code](#). Es uno de los plugins que os recomendamos instalar y que usaremos mucho durante el curso.

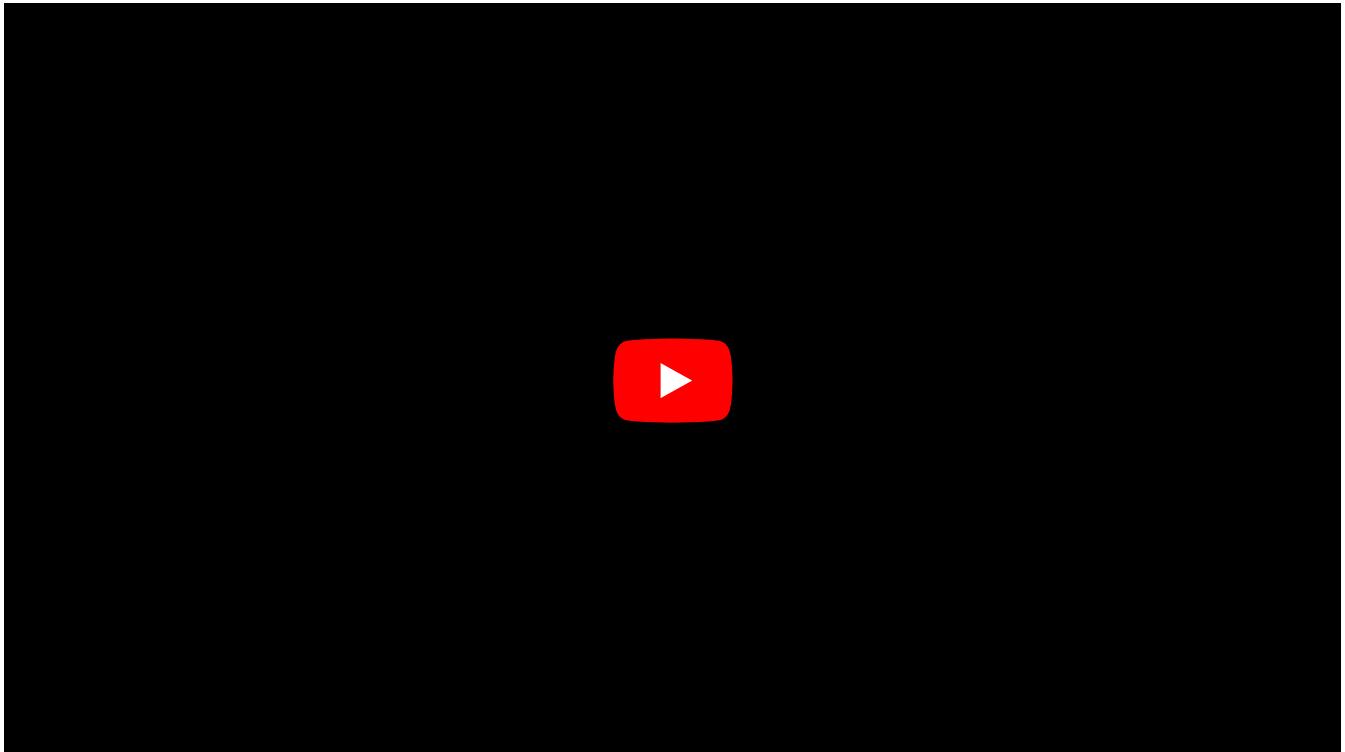
Aquí tenéis un [vídeo de cómo se usa](#):



Guía de Agile y Scrum

Agile

En contra de lo que mucha gente cree, Agile no es una metodología, es una cultura de trabajo con sus valores y sus principios, una mentalidad de cara a proporcionar un enfoque que facilite la inspección y adaptación en tiempos de incertidumbre.



Agile ([desarrollo ágil de software](#)) nace en 2001 en EEUU de la mano de varios profesionales del sector que querían dejar plasmado en un manifiesto los valores imprescindibles para el desarrollo de software de la forma más eficiente y satisfactoria posible. Merece la pena leerlo completo, tanto los valores como los principios: [Manifiesto Agile](#).

Esta fue carta presentación de Agile y lo llamaron el manifiesto para el desarrollo ágil de Software (SW). Lo que nos proponían los firmantes con este manifiesto era empezar valorar y dar más importancia a los elementos de la izquierda frente a los de la derecha. Veámoslos uno a uno:

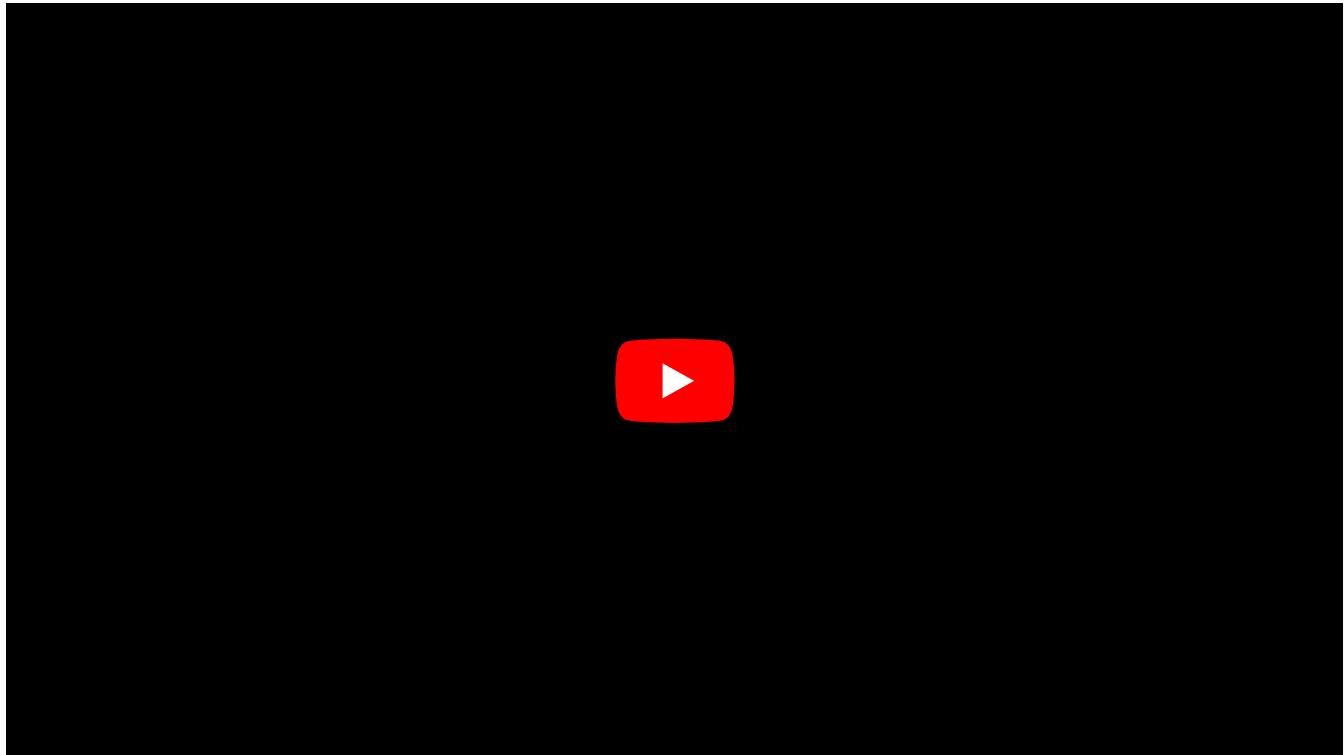
1. **Individuos e interacciones sobre procesos y herramientas:** las personas son el centro y donde la comunicación suele ser la cable, y no tanto seguir procesos predefinidos o encorsetarnos en herramientas, que es lo que haríamos si Agile fuese una metodología.
2. **SW funcionando sobre documentación:** SW funcionando cuanto antes, para fallar pronto y detectar las cosas con antelación, más allá de tener documentación exhaustiva porque siempre será mejor ver una demostración de cómo funciona algo y hacerse una idea de cómo queda que tener que leerse un manual de 200 hojas.
3. **Colaboración con el cliente sobre firmar un contrato:** colaboración con el cliente, para poder contar su feedback temprano y saber si vamos por buen camino, por encima de negociar un contrato con términos y condiciones que cumplir a largo plazo.
4. **Responder al cambio sobre seguir un plan:** adaptación al cambio frente a seguir un plan medido y permanecer impasible.

El sector tecnológico es altamente cambiante, con muchísima incertidumbre, lo que solemos llamar [VUCA](#), donde la capacidad de adaptación es crucial y la palanca del cambio se encuentra en las personas. Más allá de la complejidad de la situación que rodee la iniciativa en la que trabajemos haciendo SW, si priorizamos la visión, el entendimiento, la claridad y la agilidad como medio por encima de ese posible caos, seremos más capaces de trabajar en entornos impredecibles. Cuanta más gente colabore para buscar ese entendimiento, más fácil será encontrar soluciones, por eso Agile es una cultura que concierne a toda la organización, de forma que todo el mundo avance en el mismo sentido, con los mismos objetivos. Y, asumiendo que parte de esa complejidad no solo es inherente al contexto sino que también la añadimos las propias personas que estamos detrás del SW, será de vital importancia el desarrollo de habilidades o *soft-skills* para poder responder antes estas situaciones.

Las organizaciones ágiles y receptivas están diseñadas para aprender y responder rápidamente gracias a un feedback constante, un flujo de comunicación y colaboración clave para seguir evolucionando. En ellas se anima a experimentar y aprender en ciclos rápidos, auto-organizados en equipos, que lideran los avances y hacen crecer la empresa. Esto está directamente relacionado con los valores y los principios de la “cultura Agile” patente en el manifiesto, un mindset que muchas empresas del sector tecnológico ya han incorporado como parte de su ADN.

Resumiendo, Agile es una cultura que trasciende a toda la organización, que está descrita por los 4 valores del manifiesto y con una serie de principios que giran en torno a ellos. Todo esto se aterriza a través de un número ilimitado de prácticas como [Design Thinking](#), [Kanban](#), [eXtreme Programming \(XP\)](#) y [Scrum](#) que ayudan a su consecución.

A continuación, en el siguiente video aprovecharemos para introducir una de estas prácticas ágiles que experimentaréis en primera persona: Scrum.



Para ello, hablaremos primero del conocido [Ciclo de Deming](#), el *Plan Do Check Act (PDCA)*: planifica, haz, comprueba y actúa, y vuelta a empezar. Edward Deming fue un reputado estadista del siglo pasado que introdujo, mucho antes del nacimiento de Agile, la utilidad de trabajar de forma cíclica. Él ya se aventuraba a asegurar que trabajar mediante ciclos de inspección y adaptación era una forma eficaz de ir aprendiendo.

Y, ¿cómo funciona? Pues la idea es planificar qué hacer durante un periodo corto, a continuación comenzar hacerlo, y cuando acabe, medir qué tal ha salido para, finalmente, valorar los puntos de acción sobre el plan para ir mejorando y volver a iterar. De hecho, la teoría empírica o método científico basado en la experimentación se basa en esto. Así, el típico "prueba y error" podría corresponderse con los ciclos de Deming que se repetirían una y otra vez hasta ir alcanzando la mejor aproximación. Y aquí es donde entra en juego Scrum.

Scrum nace como un marco de prácticas ágiles orientadas al desarrollo. Scrum establece objetivos a corto plazo y, sistemáticamente, paso a paso, va encontrando la manera de llegar a ellos a través de una serie de eventos y de roles. Es lo que se llama un proceso iterativo e incremental, acorde al Ciclo de Deming anteriormente mencionado y basándose en pequeños ciclos de desarrollo, que serán cortos períodos de tiempo previamente establecido, fijo y acotado que llamaremos **Sprints**. Pero no os adelantaremos nada más... tendrás que leer la siguiente guía para profundizar en cómo vamos a hacerlo.

¿Por qué Agile en Adalab?

Introducir Agile en el momento en el momento previo a comenzar la formación es clave ya que permite dotar de información y herramientas para una mejor gestión del nuevo contexto que, como ya hemos comentado, es bastante cambiante. Y, por ello, queremos que las alumnas de Adalab entiendan el porqué de este *mindset* desde el primer día. Así, gracias a habilidades como trabajar en equipo, comunicarse de forma efectiva, la tolerancia al fallo, etc. os convertiréis en grandísimas profesionales desde el primer momento, facilitando vuestra incorporación a cualquier empresa tecnológica, comprendiendo la cultura organizativa y adaptándose al nuevo entorno de forma más eficiente, lo cual marcará la diferencia respecto a otros perfiles juniors.

Además, el propio Adalab se centra en esa cultura de empresa. Prueba de ello es que la gestión de Adalab responde a este concepto: se busca la mejora constante a través del feedback de las alumnas, profesores/as, facilitadores/as, voluntarios/as, empresas colaboradoras... para ir adaptando la formación a lo realmente demandado, poniendo foco en las personas que lo hacen posible y que aportan un valor añadido para crear un bootcamp diferenciador como este.

Para saber cómo lo trabajaremos en Adalab, podéis seguir con la siguiente sección de la guía de Scrum donde lo contaremos en detalle.

Scrum

Scrum es un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor. Se basa en la teoría empírica de control de procesos. Así, el conocimiento procede de la experiencia y en poder tomar decisiones basándose en lo conocido. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo. Además, los pilares de Scrum están cimentados en la **transparencia, inspección y adaptación**, los cuales fomentan la confianza en todo el mundo. Los miembros del equipo Scrum aprenden y exploran estos valores a medida que van trabajando en los eventos de Scrum que aquí pondremos en práctica.

Scrum en detalle

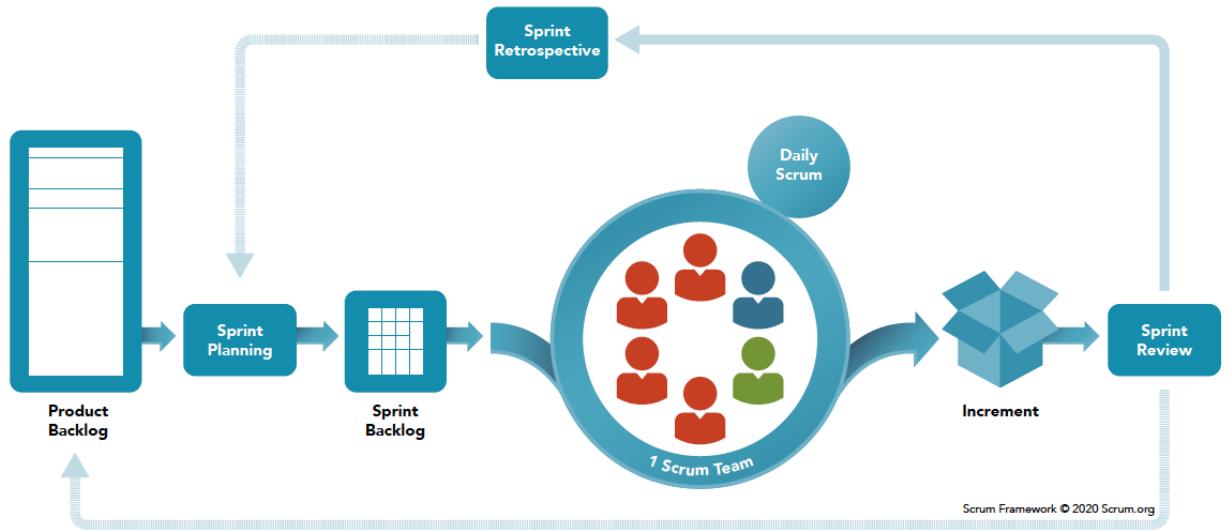
Scrum es un marco de trabajo o un *framework* anterior al Agile Manifiesto. Su palabra viene de la "melé", un tipo de formación que se realiza en rugby. En la melé, los equipos avanzan de manera conjunta como una sola unidad, justo lo que define a Scrum. Se dice que el rugby nació de la experimentación y de la adaptabilidad a base de iteraciones. Es por eso que Jeff Sutherland y Ken Schwaber, los fundadores de Scrum, escogieron este nombre presentándolo oficialmente en 1995.

Scrum nace, por tanto, como un marco de prácticas ágiles orientadas al desarrollo. Scrum establece objetivos a corto plazo y, sistemáticamente, paso a paso, va encontrando la manera de llegar a ellos a través de una serie de eventos y de roles. Es lo que se llama un proceso iterativo e incremental, basándose en pequeños ciclos de desarrollo que llamaremos sprints.

Sprints

Scrum altera la idea que tenemos del tiempo gracias a los sprints. En vez de verlo como algo lineal, en cascada, se considera algo cíclico, y esta diferencia es, realmente, el punto de inflexión para poner en valor Agile. Cada sprint es una oportunidad para hacer algo totalmente nuevo, y una oportunidad para fallar pero con poco impacto, de forma que cada día tengamos posibilidad de mejora con foco en la efectividad y productividad.

SCRUM FRAMEWORK



Sprint

Para explicar los sprints, recurriremos a [la guía de Scrum](#) (cuya lectura recomendamos) donde se muestra esta imagen como representación de un Sprint, siendo este un ciclo de tiempo corto y prefijado que se repite para ir evolucionando el producto de forma iterativa e incremental, guiado por una serie de eventos que se realizan en cada sprint y que nos ayudan a mejorar la comunicación y obtener feedback temprano. Esto, además, nos permitirá responder al cambio rápidamente y accionar un mecanismo de mejora continua más que necesario en cualquier organización.

Para accionar los sprints, el marco de trabajo de Scrum se compone del equipo Scrum (o Scrum Team) y sus roles, los eventos y los artefactos relacionados. Cada componente dentro de este marco de trabajo tiene un propósito específico y es esencial para el éxito de Scrum. El Sprint será el artefacto central de Scrum, además del Product Backlog y del Incremento de Producto. En el siguiente vídeo se revisan todos estos conceptos de una forma muy visual y en las siguientes secciones tenéis el esquema para terminar de tener la foto completa.



Roles

Los **Scrum Teams** (o equipo completo que trabaja en el producto) son auto-organizados y multifuncionales.

- Al ser auto-organizado, las personas que lo forman eligen la mejor opción de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo.
- Al ser multifuncional, el equipo ha de contar con todas las competencias y habilidades necesarias para llevar a cabo el trabajo sin depender de otras personas que no formen parte del equipo.

Este modelo de equipo está diseñado para optimizar la flexibilidad, la creatividad y la productividad. Los Scrum Teams entregan productos de forma iterativa e incremental, maximizando las oportunidades con el aprendizaje de cada iteración/sprint.

Scrum Team consta de tres **roles** esenciales: Scrum Master, Product Owner (Propietario del Producto) y Development Team (Equipo de Desarrollo).

Scrum Master

Scrum Master es la persona responsable de promocionar y explicar Scrum tal y como se define en [la guía de Scrum](#). Ejerce de líder al servicio del equipo. Ayuda a toda la organización (es decir, incluso a personas externas al equipo) a modificar las interacciones con el mismo para maximizar el valor creado por todo el equipo (Scrum Team). De esta forma, este rol ayuda a:

- Asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todo el equipo
- Encontrar técnicas para gestionar el Product Backlog de manera efectiva
- Entender la necesidad de contar con elementos en el Product Backlog claros y concisos
- Asegurar que el Product Owner conozca cómo ordenar el Product Backlog para maximizar el valor
- Facilitar los eventos de Scrum según se requiera o se necesite
- Guiar al Development Team en ser auto-organizado y multifuncional
- Ayudar al Development Team a crear productos de alto valor
- Eliminar impedimentos para el progreso del Development Team
- Entender y practicar la agilidad
- Motivar cambios que incrementen la productividad del Scrum Team y, con otros Scrum Masters para incrementar la efectividad de la aplicación de Scrum en la organización.

Product Owner

Product Owner es la persona responsable de maximizar el valor del producto del trabajo del Development Team. Es el único rol que debería gestionar el Product Backlog (toda la pila de requisitos de lo que necesita el cliente/usuario) y, por tanto, la persona encargada de:

- Expresar claramente los elementos del Product Backlog
- Ordenar los elementos en el Product Backlog para alcanzar los objetivos y las misiones de la mejor manera posible
- Optimizar el valor del trabajo que realiza el Development Team
- Asegurar que el Product Backlog sea visible, transparente y clara para que el Development Team lo entienda
- Nadie puede pedir al Development Team que trabaje en un conjunto diferente de requisitos.

Development Team

Development Team o equipo de desarrollo se compone de profesionales que realizan el trabajo de entregar un Incremento de Producto "Terminado" (*Done*) que potencialmente se pueda poner en producción al final de cada Sprint. Un Incremento de Producto "Terminado" es obligatorio en el Sprint Review (los eventos están referenciados más abajo). Los Development Team tienen las siguientes características:

- Son auto-organizados: nadie (ni siquiera el rol de Scrum Master) indica al Development Team cómo convertir elementos del Product Backlog en Incrementos de funcionalidad potencialmente desplegables
- Los Development Teams son multifuncionales, con todas las habilidades necesarias para crear un Incremento de producto
- Scrum no reconoce títulos para los miembros de un Development Team, independientemente del trabajo que realice cada persona
- Scrum no reconoce sub-equipos en los Equipos de Desarrollo, no importan los dominios particulares que requieran tenerse en cuenta, como pruebas, arquitectura, operaciones, o análisis de negocio
- Los miembros individuales del Development Team pueden tener habilidades especializadas y áreas en las que estén más enfocados, si bien la responsabilidad recae en todo el equipo.

Eventos

Scrum recomienda cuatro **eventos** (también llamados ceremonias reuniones formales), contenidos dentro del propio Sprint de cara a conseguir la inspección y adaptación necesarias para ir iterando.

NOTA: Es importante que veáis los siguientes vídeos antes de comenzar vuestro primer sprint para saber en qué consiste.

Sprint Planning (Planificación del Sprint)

Sprint Planning es el evento por el que empieza cualquier sprint. La planificación es necesaria para entender qué hacer en base a los requisitos del cliente o, mejor, a las Historias de Usuario (Product Backlog) que explican la necesidad que se quiere cubrir con la solución. De entre aquello identificado por el Product Owner como más prioritario, se estimará cuántas Historias de Usuario se pueden cubrir en un sprint (lo que llamaremos Sprint Backlog). Para valorar cómo hacerlo, se crearán tareas de desarrollo que se acordarán y repartirán entre todo el equipo con el fin de alcanzar el objetivo del sprint que va a comenzar.

En este vídeo tenéis todo el detalle.



Daily (Scrum Diario)

Daily o reunión diaria es una sincronización que tendrá lugar todos los días del sprint, a la misma hora y donde todo el equipo de desarrollo actualizará el progreso y siguiente pasos y, sobre todo, se levantarán los impedimentos que hagan peligrar el objetivo del sprint. Justo en el vídeo anterior también se introduce este evento.

Sprint Review (Revisión del Sprint)

Sprint Review es el evento que tiene lugar al final del sprint, cuando el trabajo haya concluido, y supone una revisión del Incremento de Producto que se entrega mediante la demostración del software construido funcionando. Este punto es clave, enseñarlo en directo y demostrar su valor a la audiencia para recibir feedback. Mucha gente lo llama demo, pero va mucho más allá.

Los dos siguientes vídeos están dedicados a explicar en profundidad la importancia del Sprint Review y las habilidades y actitudes necesarias (comunicación y creatividad entre otras) para su buena consecución. Recordad, es un evento necesario para revisar lo que ha hecho y chequear si era lo esperado. Todos los Sprint Reviews que experimentaréis os ayudarán a crear mejores productos e increíbles presentaciones de los mismos.



Sprint Retrospective (Retrospectiva del Sprint)

Sprint Retrospective es, en último lugar, el evento que cierra el sprint. Esta retrospectiva nos invita a reflexionar, por parte de todo el equipo, sobre lo hecho y el feedback recogido para que, desde vuestras propias ideas, proponer vías de mejoras y seguir iterando. Para ello, es muy útil saber cómo dar y recibir feedback, un proceso increíblemente enriquecedor en el ámbito profesional y en el trabajo de equipo.

En estos dos últimos vídeos, se explica con detalle la relevancia de la retrospectiva y del feedback individual y en conjunto para crecer como profesional y como parte de una empresa.



¿Por qué Scrum en Adalab?

En Adalab, **Scrum** es el marco de trabajo ágil a la hora de organizar el trabajo en los proyectos y las actividades del curso. Como ya hemos visto, Scrum nos propone una serie de buenas prácticas para organizarnos como equipo de una manera eficiente y recoger el feedback necesario para la mejora del producto, fomentando siempre la colaboración entre todas las partes y la entrega continua de software funcionando que permite aprender a pasos agigantados.

En cada uno de los módulos en los que está dividido el curso, se trabajará en equipo para desarrollar un proyecto formulado bajo el marco de Scrum y cada uno supone un reto diferente que aporta aprendizaje y experiencia.

Los 3 primeros módulos están divididos en 2 sprints que cuentan con sus correspondientes sesiones iniciales de **Sprint Planning**, a través de los cuales se discute el alcance del Sprint en base a unos requisitos de negocio en forma de Historias de Usuario. Tras eso, definen sus **Working Agreements**, que les permitirá tomar decisiones como un equipo auto-organizado o auto-gestionado. Una vez definidos, comienzan a trabajar en el sprint, realizando **Daily** diariamente como punto de sincronización donde las alumnas se ponen al día de sus avances, escollos y tareas pendientes, lo que permite que avancen como equipo de forma cohesionada e informada. Al final de cada sprint se realiza un **Sprint Review** en formato demostración donde se enseña el software funcionando, seguido de una **Retrospective** posterior. Con estas dinámicas, las alumnas aprenden a testar cada poco tiempo las soluciones propuestas y ver si son las mejores o necesitan reorientar el enfoque. Al finalizar el módulo, las alumnas presentan su proyecto ante una audiencia más amplia para verificar el resultado final y realizar una retrospectiva global del proyecto, en la que valoran de nuevo su trabajo en conjunto con sus compañeras de equipo, además del feedback recibido durante las demos, y determinan las cuestiones que facilitaron o dificultaron el trabajo para mejorar en el futuro. En el siguiente módulo, los equipos cambian para tener la oportunidad de trabajar con todo tipo de perfiles.

En cada equipo y por sprint, una de las alumnas asume el rol de **Scrum Master** y, con la ayuda del equipo docente, sirve de guía para sus compañeras en cuanto a la implementación del marco de trabajo de cara a: dinamizar los eventos, promover la consecución de los Working Agreements, ayudar en los impedimentos y, en general, acompañar al equipo en cuanto a la búsqueda de la eficiencia a lo largo del sprint. El objetivo es entender la importancia de adoptar una mentalidad de mejora continua con la capacidad de adaptarse a los cambios, de interiorizar nuevos conceptos mediante la práctica y de trabajar siempre manteniendo la vista puesta en el usuario y las necesidades que se quieren cubrir.

Y aquí comienza la aventura con vosotras de protagonistas: ya tenéis el vehículo para conseguirlo, ahora solo tenéis que empezar a conducirlo. ¡Esperamos que disfrutéis del viaje!

Preguntas Frecuentes

Sobre el Sprint Review

1. ¿Que debo hacer para el sprint review?

Durante el sprint review das/recibes feedback sobre el producto presentado, el objetivo más importante de esta sesión es presentar y mostrar el trabajo realizado: **es imprescindible hacer una demo en vivo del producto** y solo dispones de cinco minutos para la presentación. Esta sesión no es necesaria prepararla, no hay que hacer presentaciones, no es demasiado formal, es algo corto y sencillo.

2. ¿Se presenta el objetivo del sprint en cada sprint review?

Es imprescindible mostrar el objetivo del sprint, para medir si el mismo fue alcanzado, y en caso de que no se haya cumplido el objetivo plantear las nuevas tareas.

3. ¿En un sprint review deben hablar todas las integrantes del equipo?

Cada equipo de trabajo elige como organizarse, aunque lo ideal es que hablen todas. En la demo final si deben hablar todos los miembros del equipo.

4. ¿A las sprint review van los clientes o sólo personal técnico?

Estarán varios equipos de desarrollo y normalmente estará alguien de parte de los stakeholders (Product Owner o parte del propio cliente). Puede ser que hasta participe el CEO para ver los progresos y por donde va el proyecto.

5. ¿Puede servir para que todas las partes de un proyecto (front, back, devops, SEO, marketing) se pongan al día?

Puede servir para eso, pero la razón principal de un Sprint Review es mostrar el trabajo hecho y recoger feedback.

6. ¿En qué se diferencia la demo y el sprint review?

En un sprint review solo muestran la evolución del producto, mientras que en una demo final además se vende el producto, se venden ustedes mismas como programadoras: se pueden hacer storytelling, role plays, presentaciones, juegos, pueden ser todo lo creativas que queráis.

Sobre la Scrum Máster

1. ¿Se puede enumerar las tareas que puede hacer una scrum máster?

- Facilitar ceremonias: Convocar las reuniones y momentos que indica Scrum (daily, sprint review, retrospective)
- Desbloquear los impedimentos que bloquean al equipo de desarrollo. Los impedimentos pueden ser problemas técnicos, conflictos entre miembros del equipo, lidiar con impedimentos mayores (como una pandemia...). Si no sabéis como desbloquear una situación pueden recopilar lo que surja, reunir a las compañeras, recopilar información de lo que pasa y pedir ayuda (en este caso, a las profes).
- Hacer de coach al equipo de desarrollo para promover el auto-organización.
- Ayudar al equipo de desarrollo para crear productos de muy alto valor (valor: que aporte más dinero al cliente, o más visitas, o menos gastos, ...).

2. ¿Es la scrum máster la jefa?

No es la jefa, todos los miembros del equipo de desarrollo son los responsables de las decisiones.

3. ¿Es la scrum máster la que se comunique o convoque a la product owner? ¿Es el puente de comunicación con otros equipos (diseño, backend)?

En la empresa, si eso sirve para desbloquear y facilitar las comunicaciones inter-equipo, la scrum master puede realizar esa función, sí, sobre todo para no estar molestando al equipo con esas funciones y que esté dedicado a ser productivo.

Si la comunicación entre profes y equipos no es fluida (normalmente suele salir de forma orgánica), la Scrum Master puede levantar la situación y que se valore entre todas quién hace ese rol comunicativo o, si nadie quiere, para que no exista bloqueo, que sea la Scrum Máster quien lo haga.

4. ¿Debe vigilar que todo el equipo siga las pautas de Scrum (que muevan las tarjetas de columna, que desglosen las user story)?

No lo llamamos vigilar, la Scrum Máster solo tiene que hacer entender al resto de compañeras que mover las tarjetas de columnas es útil para estar sincronizadas, o que desglosar las user stories les viene bien para dividirse el trabajo y detallar los pasos. Es velar porque haya un mínimo de organización para trabajar en equipo.

Guía para enfrentarse a problemas técnicos

En nuestro día a día las programadoras nos enfrentamos a problemas que, muchas veces, nos resulta difícil solucionar solas.

Para Adalab y para los profes es importante que sepas:

- **A programar se aprende programando**, probando cosas, **rompiendo cosas** que ya funcionan para ver qué pasa, **solucionando errores...**
- Saber solucionar errores es más importante que saber hacer un código bien a la primera. **Nadie consigue hacer un código bien a la primera**. Sería como pintar un cuadro sin fallar ni una sola pincelada.
- Los errores no se pueden enseñar. **A solucionar errores se aprende solucionándolos**.
- **Al solucionar un error aprendemos muchísimo** sobre cómo funciona la programación.
- **Todos las dudas, problemas y errores que tengas ya los ha tenido alguien antes**. Y lo más probable es que esa persona haya preguntado en un foro, escrito un artículo o subido la solución a un repo...
- La programación evoluciona muy rápido, debemos saber buscar recursos para solucionar errores y estar al día.
- Durante toda tu vida como programadora te enfrentarás a muchísimos errores.
- **Todo programador**, por muy senior que sea, **siempre es junior en algo**.
- Las profesoras te ayudaremos mucho durante el curso, pero después tendrás que ser autónoma.
- Si las profes tuviéramos que elegir una sola cosa que enseñarte bien, sería **a ser autónoma, autodidacta y saber buscar tus propios recursos para seguir aprendiendo**.

Afortunadamente existen montones de recursos en los que apoyarnos. Os recomendamos:

Páginas famosas con recursos técnicos

- Resolución de dudas y búsqueda de errores: [Stack Overflow](#).
- Documentación resumida SQL: [w3schools - SQL](#).
- Documentación resumida Python: [w3schools - Python](#).

Documentación oficial

Una buena idea es consultar la documentación oficial de los lenguajes y herramientas que estés usando:

- [Documentación oficial para Python 3.9](#)
- [Documentación oficial para MySQL](#)

Puede ocurrir que la documentación no cubra nuestra duda o que el lenguaje que use aún nos resulte

demasiado tecnico, en cuyo caso es momento de buscar en Google.

Búsqueda en Google

Buscar en Google nuestro problema es muy buena opción y aunque no hace falta redactar la pregunta como la haría William (Shakespeare) sí que usar el inglés nos va a dar una ventaja significativa.

Vamos a ver algunos ejemplos, cuando buscamos información y cuando buscamos un error concreto:

Buscar información

- Nos suena que el método `str.split()` nos va a solucionar la vida al trabajar con texto, pero no sabemos o no nos acordamos cómo usarlo, ni dónde estará la documentación:
 - [python string split](#)
- Estamos trabajando con listas numéricas y no sabemos como ordenarlas:
 - [python check if list is sorted](#)
- Si alguna vez deseas eliminar ciertos resultados de búsqueda, será suficiente con incluir al final de la búsqueda `-busqueda_a OMITIR`. Ejemplo: Realizar un histograma en python sin usar matplotlib
 - [python plot histogram -matplotlib](#)

Buscar un error concreto

A veces nos van a saltar errores que podríamos no entender. Una opción es copiar el error, meterlo entre comillas, añadir el lenguaje de programación para ayudar a Google a acotar las respuestas y buscar a ver qué sale.

- En este ejemplo hemos intentado obtener un valor de una lista y en la consola nos aparece el error `IndexError: list index out of range`. Vamos a suponer que no entendemos que pasa y nos vamos a nuestro buscador favorito a buscar:
 - [python "IndexError: list index out of range"](#)

En inglés

Siempre es mejor buscar en inglés, intenta no complicarte con sentencias muy elaboradas. Usa palabras claves sencillas y lo más probable es que Google te sugiera autocompletar la búsqueda en base a lo que han buscado otras personas.

Resultados

Una vez hecha la búsqueda lo mejor es comenzar con los resultados de páginas que ya conocemos y en las que confiamos como **Stack Overflow** o **Geeksforgeeks**. Si no aparece ninguna de estas, pues como en cualquier búsqueda empezamos con el primer resultado

Puede que entre los resultados de Google aparezca [Stack Overflow](#). Es un foro muy, muy, muy útil para compartir conocimientos y resolver errores. Es tan famosa que a veces, cuando una empresa va a contratar a un programador, mira su reputación en Stack Overflow.

Lo más probable es que alguien ya haya realizado la misma pregunta o una muy similar y que la comunidad le haya ayudado a resolverla. Veamos lo básico de Stack Overflow.

Supongamos que estamos programando y no recordamos claramente la diferencia entre loc e iloc, en la librería de Python Pandas . Lo buscamos en Google `pandas loc and iloc different` y como primer resultado nos sugiere [esta página de Stack Overflow](#).

Ábrela y observa los números y flechas que hay a la izquierda de la pregunta y las respuestas.

Label vs. Location

 1250 The main distinction between the two methods is:

-  • `loc` gets rows (and/or columns) with particular **labels**.
-  • `iloc` gets rows (and/or columns) at integer **locations**.

 +150 To demonstrate, consider a series `s` of characters with a non-monotonic integer index:

```
>>> s = pd.Series(list("abcdef"), index=[49, 48, 47, 0, 1, 2])
49    a
48    b
47    c
0    d
1    e
2    f

>>> s.loc[0]      # value at index label 0
'd'

>>> s.iloc[0]    # value at index location 0
'a'
```

Una respuesta en Stack Overflow

1. A la izquierda de la pregunta tenemos un número, ese número indica los "me gusta" o votos de otros programadores. Puede ser muy alto si es una buena pregunta o incluso negativo si es una pregunta que no está bien redactada, o no tiene sentido.
2. A la izquierda de las respuestas también tenemos el número de "me gusta" o votos. La respuesta con más "me gusta" es la que más nos interesa.
3. Si una respuesta tiene un check verde significa que la persona que preguntó ha marcado esa respuesta como válida, ya que le ayudó a solucionar el problema.
4. Muchas veces una pregunta viene marcada como duplicada, con un enlace a otra pregunta muy similar.

Comunidad

Cuando todo lo demás ha fallado, o hemos encontrado una solución pero no estamos del todo conformes con ella hay que recurrir a la comunidad más cercana, preguntando:

1. A compañeras de curso.
2. En canales de Slack.
3. A las profesoras.

Guía para pair programming

En *pair programming* dos personas se enfocan en desarrollar una aplicación, como programadora puedes decidir trabajar con otra programadora para encontrar una solución más rápida y obtener mejores resultados. Se puede utilizar esta técnica en diferentes contextos, es una técnica de desarrollo ágil de software muy valorada en las empresas de desarrollo digital y que estaremos practicando durante todo el curso de Adalab.

Para las sesiones de *pair programming* en Adalab nos planteamos como reto realizar una serie de ejercicios propuestos de acorde al tema de la clase de ese día. Este ejercicio en algunos casos serán independientes y en otros será continuado e iremos añadiendo características y funcionalidades poco a poco.

En *pair programming* dos personas desarrollan juntas, cada una con un rol. Existen diferentes técnicas y estilos para aplicar pair programing, y va a depender de diferentes factores como el contexto y el nivel de las programadoras. Os proponemos utilizar alguna de las dos técnicas siguientes.

Técnicas y estilos del pair programming

Driver-Navigator

En esta técnica hay dos programadoras con dos roles:

- **Conductora o _driver_** la programadora se encarga de escribir el código y enfocarse en los pequeños detalles. Todos los detalles de implementación son su responsabilidad, por ejemplo tipo y nombre de las variables, métodos, funciones, clases. Explica en todo momento que está haciendo, con el propósito de explicar cuál es su intención.
- **Copiloto o _navigator_** está observando al *driver* mientras escribe el código y se enfoca en las decisiones a largo plazo, como posibles alternativas, posibles fallos del código.

Nota: el rol de la copiloto es clave en esta técnica de Pair Programming, tiene que estar activa guiando a la compañera: compartiendo ideas y conocimiento, indicando pasos a seguir y errores, proponiendo alternativas más optimas... Para que la conductora se pueda centrar en escribir. ¡Nada de desconectar mirando el móvil!

Configuración En esta técnica la conductora es la que escribe el código, el copiloto no es necesario que intervenga. El copiloto se encarga de buscar recursos, buscar respuestas, mirar la documentación.

Rotaciones Las dos programadoras cambian de rol frecuentemente, para aumentar la eficiencia del código. Durante el curso os recomendamos cambiar de rol en cada ejercicio, o incluso a mitad de los más extensos. Ambos roles mantienen una comunicación verbal continuada y abierta.

Elastic Pairing

En este estilo de pair programing cualquier programadora puede tomar el control del teclado y continuar programando. Las decisiones son tomadas como consecuencia de una buena comunicación, donde las integrantes de la pareja saben explicar su punto de vista y sus argumentos son aceptados por la otra programadora. Las programadoras intentan encontrar la mejor solución al problema, y no intentan imponer su opinión basada en el ego.

Una buena regla de este estilo es trabajar más despacio, porque tu compañera te está mirando, y así se cometen menos errores. También es conveniente establecer reglas como ni usar el teléfono, no distracciones y así nos concentramos en el código que estamos programando.

Esta sería la mayor expresión de colaboración entre parejas, puedes trabajar en conjunto con tu compañera respetando sus puntos de vista y practicando la empatía.

Beneficios

Al principio te costará ver los beneficios del trabajo en parejas, pero tienes que confiar en nosotros y darle una oportunidad con una mente abierta, porque son muchos, por ejemplo:

- Compartir conocimientos (dar/recibir) es vital para el aprendizaje, y el desarrollo de software es una profesión con un alto reciclaje de conocimientos, donde la formación continua es parte del trabajo.
- Cuatro ojos ven más que dos, esto es muy útil para aumentar la calidad del código evitando *bugs* o errores.
- El trabajo compartido es más agradable, y la confianza en el mismo también es más alta cuando se comparte.
- Mejora las habilidades comunicativas y la empatía de las desarrolladoras, herramientas claves para el trabajo en un equipo.
- Aumenta el foco, es más difícil procrastinar cuando estamos trabajando activamente con otra persona, también desde el exterior el resto de personas tienden a interrumpir menos a dos personas hablando que a una callada.

Herramientas

Una herramienta estupenda para programar en remoto es [Live Share de VS Code](#). Es uno de los plugins que os recomendamos instalar y que usaremos mucho durante el curso.

Aquí tenéis un [vídeo de cómo se usa](#):



Organización de las sesiones

En cada sesión de pair programing os recomendamos:

1. Decidir con qué técnica o estilos de pair programing (**Driver-Navigator** o **Elastic**) se sienten más cómodas para utilizar con su pareja.
 - En caso de elegir **Driver-Navigator** alternar en cada ejercicio los roles.
2. Plantearse el objetivo del pair programing, en cada sesión resolveremos el ejercicio global que aplica lo visto en clase, pero si sientes que todavía no entiendes bien el contenido de ese día podéis acordar hacer los ejercicios de la clase. Algunos objetivos a plantearse pueden ser:
 - Hacer lo correspondiente al ejercicio global.
 - Completar los ejercicios X,Y,Z... de la clase.
 - Repasar los ejercicios resueltos de la explicación en clase.
3. Decidir con qué herramienta y de que manera vais a trabajar en remoto en pareja, por ejemplo:
 - Utilizar la extensión Live Share y compartir el proyecto.
 - Compartir pantalla y solicitar el control de la pantalla de la compañera.

Guía para la evaluaciones técnicas

En esta guia se incluye en detalle los diferentes procesos previos a las evaluaciones y algunos consejos:

- El dia previo a la evaluación:
 - A las 8.30h como fecha límite se entregará los ejercicios resueltos por las alumnas del contenido de pair programming.
 - A 15h se compartirá el orden las horas de evaluación para cada una de las parejas y con que profesora deberéis realizar la evaluación.
- El día de la evaluación:
 - Os deberéis conectar directamente a la sala de la profe asignada.
 - En la evaluación se os podrá preguntar cuestiones sobre los ejercicios entregados y conceptos teóricos que se consideren relevantes.
 - La evaluación durará unos 40 minutos por pareja.
- Despues de la evaluación:
 - Se os mandará de forma privada a cada alumna, los resultados de la evaluación, junto con algunos comentarios de aspectos a destacar y aspectos a mejorar de cara a las siguientes evaluaciones de contenido.

Consejos para las evaluaciones técnicas:

- Intentar ir lo más relajadas y tranquilas posibles, las profes no nos comemos a nadie (al menos de momento).
- Si alguna de las preguntas que os podamos realizar no la sabéis, ser honestas y decir que no lo recordáis o que deberíais mirarlo para responder de forma segura y no equivocarse. La sinceridad se valora más que intentar decir la primera cosa que pasa por la cabeza, con tal de salir del paso.
- Estáis en un entorno seguro y os queremos ayudar a crecer como personas y como profesionales a la hora de enfrentarlos a las entrevistas de trabajo técnicas una vez termine el bootcamp, así no tengáis miedo de meter la pata o cometer errores. Mejor cometer errores ahora, que el día de una entrevista real. Se os dará feedback del proceso para que cada vez se os de mejor enfrentarlos a este tipo de pruebas!

Guía para el dia final del módulo

En esta sección os queremos recopilar otras guías e información que os será muy útil a lo largo del curso:

Día Final de cada módulo

Es interesante que esta sesión te la leas unos días antes del último día del primer módulo. No es necesario que lo veas antes de empezar el curso.

Como sabemos, al final de cada módulo tenemos un día dedicado a la graduación del módulo en el que entre otras cosas haremos:

- Demo del proyecto
- Retro individual
- Retro del proyecto
- Feedback a Adalab (individual y World Café)

Demo del proyecto

- Para realizar la demo del proyecto debes haber visto los [vídeos 5 y 6 de Scrum](#) que van sobre Sprint Review y Demo de proyectos.
- En la primera parte de la demo cada equipo realizará una presentación de cinco minutos, esta presentación puede ser más elaborada con un Power point o similar.
- En la segunda parte de la demo se realiza el feedback de las compañeras en no más de cinco minutos. El objetivo es dar feedback como espectadora de la demo con aspectos a mejorar y aspectos a destacar. Recordar que sin feedback no hay mejora!

Retro Individual

- Para realizar la sesión de Retroespectiva debes haber visto [los videos de Scrum 7 y 8](#) que explican detalladamente la retrospectiva.
- Esta sesión tiene lugar en las mini salas de equipos y el objetivo es realizar una retro de cada una de los miembros del equipo.
- Podemos realizar 1 post-it a mejorar y 1 post-it a destacar de cada una de las compañera. Empezamos con una compañera, todas le damos feedback y después pasamos a la siguiente. Podemos hacerlo por escrito usando cualquier herramienta como el GithubProject, Word.
- El objetivo es mejorar y ver si tenemos la misma percepción de nosotras mismas es igual a las percepción que tiene sobre mi el resto de compañeras.

Retro Equipo

- Esta sesión tendrá una duración de 45 minutos, dinamizada por la Scrum Master.
- Os proponemos la dinámica del barco que plantea hacer una retro siguiendo las siguientes líneas:



Dinámica del barco para equipos

- El viento: cosas a nuestro favor.
- El ancla: dificultades en el camino.
- La isla: objetivos alcanzados, ¿coinciden?
- El ron: menciones especiales.
- 1 post it por cada apartado.
- Se puede utilizar la herramienta [Easy Retro](#)
- También se puede utilizar la dinámica de la estrella (os recomendamos usarla para el próximo módulo)



Feedback a Adalab

Tu feedback sobre Adalab y la formación es lo que nos permite mejorar continuamente. Al final de cada módulo, haremos una dinámica individual y una dinámica colectiva para recopilar ese feedback:

Feedback Individual

Después de las retrospectivas te pasaremos un cuestionario anónimo para que nos puedas dar tu feedback individual sobre el módulo: los materiales, las profesoras, la metodología, y otros aspectos.

World Café - Feedback colectivo

El penúltimo día de cada módulo las SCRUM Master deben recoger el feedback de su equipo sobre Adalab y juntarse para poner en común las conclusiones de todos los equipos. El día final de cada módulo dos voluntarias nos pasarán este feedback durante la sesión, y haremos preguntas en caso de querer aterrizar algunas cuestiones. Para el feedback, la estructura es la siguiente:

Materiales y recursos del curso

Materiales, ejercicios, herramientas, etc.

- 3 aspectos a destacar (cosas que están muy bien y se deben mantener/reforzar)
- 3 aspectos a mejorar (cosas que se pueden mejorar o cambiar)

La formación en Adalab

La docencia, sesiones de desarrollo profesional, Kahoot, evaluaciones, tutorías, trabajo en parejas, otras actividades, etc.

- 3 aspectos a destacar (cosas que están muy bien y se deben mantener/reforzar)
- 3 aspectos a mejorar (cosas que se pueden mejorar o cambiar)

¿Qué debes hacer antes del día final de cada módulo?

La semana anterior

1. Ver los vídeos 5 a 8 de SCRUM
2. Preparar la demo y su presentación con tu equipo
3. Preparar la dinámica de la retrospectiva grupal (si eres Scrum Master)

El día anterior

1. Preparar los post-its para cada uno de los puntos de la retrospectiva (todas)
2. Preparar el feedback individual que daremos a cada compañera y a nosotras mismas (1 aspecto a destacar y 1 aspecto a mejorar)
3. Preparar el feedback a Adalab: entre todas, definir 3 aspectos a destacar y 3 aspectos a mejorar de los materiales y recursos y de la formación.

Módulo 1: Python Básico y SQL Básico

Python

Contenido

La distribución de qué vemos en qué clase.

1

Variables, booleanas, operadores numéricos y binarios

¿Qué es una variable? ¿Qué tipos hay? ¿Qué es un método? Miramos operaciones algebraicas y binarias, y introducimos unos comandos útiles.

- `isinstance()`, `type()`
- `+ - * / ** // %`
- `== != <= < >= >` and or is not
- `print()`, `round()`, `input()`
- `<Tab>`

2

Listas 1: Listas enteras predefinidas

¿Qué es una lista? ¿Cómo se crea? ¿Qué información puedo obtener sobre una lista? Miramos propiedades de la lista y comprobamos si la lista contiene un dato sí o no, y cuantas veces.

- `[]`, `list()`
- `+`
- `len()`, `min()`, `max()`
- `in`, `not in`
- `.clear()`, `.copy()`, `.count()`, `.sort()`, `.reverse()`

3

Listas 2: Modificar listas

Juntar listas predefinidas. Añadir información a una lista. Sacar ciertos datos de la lista. Practicamos con los índices de la lista.

- `[], [-1], [-1:1], [x], [-x], [x]`
- `.index()`
- `del`
- `.append()`, `.extend()`
- `.insert()`, `.pop()`, `.remove()`
- `sorted()`

4

Diccionarios

Alternativa forma de juntar datos, en que cada dato tiene una clave que lo identifique. Manipular diccionarios predefinidos, y modificar contenidos de un diccionario.

- `{}, dict()`
- `l.update()`
- `len()`, `in`, `not in`
- `.keys()`, `.values()`, `.items()`
- `.clear()`, `.copy()`, `.pop()`, `del`
- `.get()`, `.popitem()`

Repaso

5

Tuplas

Un contenedor para múltiples valores, parecido a listas y diccionarios, pero inmutable. ¿Para qué nos sirven? ¿Qué puedo hacer con ellas?

- `(), tuple()`
- tuplas son inmutables (¿qué es?)
- `len()`, `in`, `not in`, `+, .min()`, `.max()`, `.index()`, `.count()`, `(x,y)`

6

Sets

El último tipo de contenedor, que no permite duplicados y no tiene índices. ¿En qué situaciones nos interesa? Vemos las diferencias entre los cuatro tipos de contenedores.

- `(), set()`
- `len()`, `in`, `not in`
- `.copy()`, `.add()`, `.remove()`, `.clear()`
- `union`, `subset`, `superset`, `difference`, `intersection`, `<, >, ^, -, |, &`

7

Loops 1

Ejecutar código sólo si cumpla una(s) condición(es). Flujo del código: no siempre es de arriba a abajo. Cómo manipular el flujo de ejecución.

- `if: elif: else`
- `while: break`
- `try: except: else: finally: continue`

8

Loops 2

Usar contenedores (listas, diccionarios, tuplas, sets) para definir las condiciones para un loop. Miramos cómo codificar los loops de forma comprimida.

- `for: continue`
- `break`, `else: pass`
- `range`
- `comprehensions`

Repaso

Resumen gráfico de las primeras dos semanas

Las primeras dos semanas intentamos daros una base de Python. ¿Es mucho? ¡Definitivamente!

No os asustéis, siempre que perdéis el hilo las instructores estamos aquí para ayudarlos a reencontrar el camino. Piénsalo así: no puedes aprender a leer sin conocer por lo menos algunas de las letras, y encima las letras se parecen mucho entre sí. Con el tiempo interiorizas la forma de cada una y como suena, y al final ni te das cuenta de las letras que ves, sólo la información de la frase que escriben. Pues con programar es parecido: todos los paréntesis, doble puntos, y colores del código parecen ser muy complicados. Al final de estas dos semanas, verás que son muy útiles todas esas cosas, ¡y que tú también sabes leerlas!

Introducción

```
1 import os, time
2 x = "Welcome to The Mirror."
3 y = 0
4
5 while y <= len(x):
6     os.system("clear")
7     print(x[:y])
8     time.sleep(0.2)
9     y = y+1
10    time.sleep(2)
11 x = "You will stay here for as long as you can, you may
12 forfeit when you would like to."
13 y = 0
14
15 while y <= len(x):
16     os.system("clear")
17     print(x[:y])
18     time.sleep(0.2)
19     y = y+1
20    time.sleep(2)
21 x = "Please stand here, and stare into the mirror."
22 y = 0
23
24 while y <= len(x):
25     os.system("clear")
26     print(x[:y])
27     time.sleep(0.2)
28     y = y+1
29    time.sleep(2)
30 x = "This is you in the mirror:"
31 y = 0
32
33 while y <= len(x):
34     os.system("clear")
35     print(x[:y])
```

Ejemplo de un código Python en VS Code.

En la imagen arriba tenéis un ejemplo de un código en Visual Studio Code. ¿Qué cosas ya podéis identificar? ¿Qué aspecto os resalta más?

A la izquierda hay números indicando las líneas. También notáis unos cuantos colores, e una indentación variable. En realidad todos estos aspectos son ayudas para que nosotras, las humanas, lo puedan leer con más facilidad. Python en específico es un lenguaje de programación que fue diseñado con ese aspecto en mente: ser más legible.

- ⓘ La documentación de Python es excelente, échala un ojo siempre y cuando no os acordáis de cómo era alguna función o algún método:

<https://docs.python.org/3/index.html>

Por ahora vamos a empezar con los variables y algunos operadores.

Variables, booleanas, operadores numéricos y binarios

¿Qué es una variable? ¿Qué tipos hay? ¿Qué es un método? Miramos operaciones algebraicas y binarias, y introducimos unos comandos útiles.

Variables, booleanas, operadores numéricos y binarios

La primera clase de programación Python se enfoca en variables. Los vamos a mirar usando unos jupyter notebook, que nos permite dar información entre bloques de código. Todo lo que programamos en los Jupyter, también se puede escribir en VS Code. Una **variable** es como si le dieras al ordenador una tarjeta con un contenido de cualquier tipo, diciéndole "Guárdame esto bajo un nombre específico." La forma en decírselo es `nombre = contenido`.

En la matemática nos enseñaron que lo que va delante y detrás del símbolo `=` son intercambiables, es decir que son sinónimos. En la programación, sin embargo, no funciona así. El nombre se guarda como una variable, cuyo contenido puede *variar*. El contenido que le asignas se puede cambiar, o sobrescribir, repitiendo la misma asignación que la primera vez. Sería como darle al ordenador otra tarjeta diciéndole "Tira la anterior, que he cambiado de opinión."

Ojo que el ordenador no sabe qué responderte si le preguntas por el `contenido` de la variable, sólo su `nombre`. Las tarjetas que se le damos no las mira, las guarda sin saber que contienen. A la hora de querer utilizar ese contenido le preguntamos por el nombre de la tarjeta, y la devuelve. Así puede ir guardando un montón de contenidos de cualquier tipo sin necesitar que estructurarlos, sólo memorizar los nombres que las asignaste.

En la vida real ya ves a millones de variables cada día sin darte cuenta. Todo lo digital que no sea siempre lo mismo, es decir que pueda variar, estará guardado en una variable. Tu nombre de usuario por ejemplo, lo ves en muchos sitios en tu ordenador. En todos esos sitios, el ordenador mira el contenido de la variable usuario y lo demuestra ¡sin que te enteras de que haya sido una variable!

Variables entonces son la base de la programación. En esta clase miramos algunos tipos y acciones básicas que puedes hacer con ellas.

¡A por ello!

Jupyter Notebook

 modulo-1-leccion-01-variables.ipynb 82KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Reparo

Tipos de variables

- *String*: cadena de texto
- *Float*: número decimal, puede ser negativo o positivo
- *Integer*: número entero, puede ser negativo o positivo
- *Booleanos*: True o False

`type()` : nos permite saber de que tipo es nuestra variable

`isinstance` : nos permite evaluar si una variable es de un tipo especificado

```
buenos_dias = "Buenos dias Adalabers"  
# es la variable buenos_dias de tipo float?  
isinstance(buenos_dias, float)
```

- `input()` : nos permite preguntar al usuario. POR DEFECTO NOS DEVUELVE UN STRING.

Operaciones matemáticas

- `+`, suma
- `-`, resta
- `*`, multiplicación
- `**`, elevar
- `/`, división
- `//`, división entera
- `%` módulo o resto de la división

`round` que nos permite redondear los decimales.

Operaciones lógicas / comparación

- `>=`: mayor que / mayor o igual que
- `< (=<)`: menor que / menor o igual que
- `==`: igual a
- `!=`: distinto a
- `is`: nos compara si son iguales, pero mira el DNI
- `is not`: nos compara si son distintas, pero mira el DNI

Reparo

Tipos de variables

- `String`: cadena de texto
- `Float`: número decimal, puede ser negativo o positivo
- `Integer`: número entero, puede ser negativo o positivo
- `Booleanos`: True o False

`type()` : nos permite saber de que tipo es nuestra variable

`isinstance` : nos permite evaluar si una variable es de un tipo especificado

```
buenos_dias = "Buenos dias Adalabers"  
# es la variable buenos_dias de tipo float?  
isinstance(buenos_dias, float)
```

- `input()` : nos permite preguntar al usuario. POR DEFECTO NOS DEVUELVE UN STRING.

Operaciones matemáticas

- `+`, suma
- `-`, resta
- `*`, multiplicación
- `**`, elevar
- `/`, división
- `//`, división entera
- `%` módulo o resto de la división

`round` que nos permite redondear los decimales.

Operaciones lógicas / comparación

- `(>=)`: mayor que / mayor o igual que
- `<(<=)`: menor que / menor o igual que
- `==`: igual a
- `!=`: distinto a
- `is`: nos compara si son iguales, pero mira el DNI
- `is not`: nos compara si son distintas, pero mira el DNI

Pair programming

Antes de empezar

- Git repo
- Estructura de carpetas
- Documentación / readme
- Como es la evaluación

Pair programming variables

En la lección hemos hecho una primera aproximación a Python, hemos aprendido que tipos de datos tenemos, como podemos almacenar información en variables y como realizar ciertas operaciones con ellas.

Además hemos aprendido que las variables de tipo *string* o de texto tienen muchos métodos propios que nos permiten modificarlos.

Los objetivos del *pair programming* de hoy son:

- Cread tres variables numéricas de tipo *integer* (entero), llamadas de la siguiente forma:
 - numero1
 - numero2
 - numero3
- Utilizando los operadores lógicos:
- En vuestro caso, es el numero1 mayor que el numero2
 - Es el numero1 menor que el numero2 pero mayor que el numero3
 - Es el numero3 igual que el numero2
 - Es el numero2 distinto que el numero1
- Cread dos variables numéricas de tipo *float* (decimal) con dos decimales, llamadas de la siguiente forma:
 - altura1
 - altura2
- Utilizando los operadores matemáticos aprendidos en la lección:
- Cuál es la suma de las dos alturas
 - Cuál es la diferencia entre las alturas
 - Cuál es el resto de la división de las dos alturas
 - Cuál es resultado de la división de las dos alturas, redondead el resultado a un decimal.
- Escribid un programa usando la función `input()` que le pregunte al usuario su nombre y apellidos.
Una vez que tengamos el nombre del usuario:

- Printead su nombre y apellidos todo en mayúsculas
 - Printead su nombre y apellidos todo en minúsculas
 - Printead su nombre y apellidos con la primera letra del nombre y los apellidos en mayúscula y el resto en minúscula.
 - Printeas solo la primera letra del nombre en mayúscula.
- Escribid un programa usando la función `input()` que:
 - Le pregunte al usuario por una frase, almacenad este resultado en una variable.
 - Le pregunte al usuario por una vocal, almacenad este resultado en una variable.
 - Printead la misma frase pero con la vocal introducida en mayúsculas.
- Escribid un programa usando la función `input()` que:
 - Le pregunte al usuario por la lista de la compra, separando cada elemento por comas.
 - Devolved la lista de compra donde veamos cada elemento en una línea.

Pista

- Para indicar salto de línea usaremos "`\n`".
- Deberéis usar el método `join` de los *strings*

Output deseado

```
resultado_del_input = ["peras", "manzanas", "naranjas"]
```

```
# output después del ejercicio  
"peras"  
"manzanas"  
"naranjas"
```

■ Happy coding ■

Listas 1

¿Qué es una lista? ¿Cómo se crea? ¿Qué información puedo obtener sobre una lista? Miramos propiedades de la lista y comprobamos si la lista contiene un dato sí o no, y cuantas veces.

En esta clase introducimos un tipo de contenedor para múltiples valores: la lista. Con una lista se puede agrupar datos y guardarlas en una variable. Los contenidos en sí también pueden ser variables. Aparte de que listas contienen múltiples datos, la gran diferencia con los variables vistas hasta ahora es que una lista es mutable. ¿Qué es eso? Pues que el espacio que ocupa una lista en la memoria es fija, pero que lo que haya guardado ahí no es fijo. En otras palabras, listas pueden ser manipuladas, sin redefinirlas.

Damos un paso atrás, eso de la memoria ¿de qué va? Pues que el ordenador tiene una memoria, como seguro que hayas oido hablar de RAM (memoria de acceso aleatorio) a la hora de comparar ordenadores y sus habilidades. Esa memoria la tiene disponible en cualquier momento, y su uso, como ya indica el nombre, es aleatorio. Es decir, para cada variable que quieras guardar coge un sitio aleatorio en su memoria para guardarla. Si preguntas por la variable se acuerda dónde está el contenido y lo lee. Ahora bien, si defines por ejemplo la variable `dimension_en_metros = 3.4` ese integer lo guarda en un sitio random. Si luego te das cuenta que la dimensión era un poco menos, y la cambiaras, ¿qué pasa? Al decir `dimension_en_metros = 3.2` creas una nueva instante de la misma variable. En otras palabras: te borra lo anterior, en el sitio en que estaba, y se busca un sitio aleatorio para guardar este "nuevo" dato. Los integers, entonces, son objetos inmutables.

Las listas, por otro lado, son objetos mutables. Significa que hay formas de manipular el contenido de la lista en el mismo sitio, sin que se borre y redefine. Llamando por ejemplo el método `.sort()` la lista en la memoria se ordena, sin devolver o imprimir nada. Imprimiendo la misma lista antes y después de aplicarle el método te da resultados distintos.

Para esta primera clase nos enfocamos en listas que son definidas una vez, sin tocar los contenidos todavía. Miramos como definirlas y como sacar propiedades suyas como la longitud de la lista, su valor máximo y mínimo, entre otras.

Jupyter Notebook



modulo-1-leccion-02-listas1.ipynb 37KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso de listas I

- Las listas las definimos con corchetes
- Las listas pueden contener distintos tipos de datos (*strings, integers, floats, listas, etc.*)
- Podemos crear listas vacías
- Podemos convertir un *string* en una lista, usando `list()`.
- Son mutables
- Métodos de las listas:
 - `copy()` : hacemos una copia exacta de la lista
 - `clear()` : borra el contenido de la lista, nos la sobreescribe!!! △△ Cuidado que borramos toda la info
 - `count()` : nos cuenta el número de elementos en la lista de un valor especificado
 - `sort()` : nos permite ordenar la lista. Por defecto de menor a mayor. Si queremos ordenar de mayor a menor usaremos `reverse = True`
- Propiedades de las listas:
 - `len()` : nos devuelve el número de elementos de la lista
 - `max()` : nos devuevle el valor máximo de la lista
 - `min()` : nos devuelve el valor mínimo
- `in` y `not in`: nos chequea si un valor especificado está en la lista.

- Podemos unir listas:

```

lista1 = [1,2,3]
lista2 = [4,5,6]

# si lo hacemos con comas
lista1 , lista2
[[1,2,3], [4,5,6]]


# si lo hacemos con +
lista1 + lista2
[1,2,3,4,5,6]

```

- Los elementos en las listas empiezan a númerarse en 0.
- **Recomendación**, usar corchetes para definir las listas

Pair programming

En la la lección de hoy hemos aprendido una de las estructuras de datos más usadas en Python, las listas. En el ejercicio de hoy pondremos en práctica los métodos aprendidos, vuestros objetivos serán:

- Usando la función *input*, cread un programa que pregunte al usuario por el nombre de tres amig@s.
- Almacenad los resultados del paso anterior en una lista. **Pista** Tendréis que usar métodos de los *strings* que aprendimos en la lección anterior.
- Usando la función *input*, cread un programa que ahora le pregunte al usuario por las alturas de sus amig@s. Y almacenar los resultados en una lista.

Esto os devolverá una lista de *strings* similar a esta:

```
lista_alturas = ['1.45', ' 1.67', ' 1.57']
```

Nos interesa que las alturas estén en tipo *float* para eso, ejecutad el siguiente código. No es necesario que lo entendáis, es un concepto más avanzado que aprenderemos en las siguientes lecciones.

```
lista_alturas = [float(i) for i in lista_alturas]
```

- Ordenad las dos listas de mayor a menor. ¿Lo podréis hacer para la lista de nombres?
- ¿Cuál es la altura máxima de todos nuestros amig@os?
- ¿Y la mínima?
- Dadas las siguientes listas:

```
lista1 = [44, 55, 67, 44, 98, 29]  
lista2 = [34, 56, 56, 78, 67, 56]
```

- Unid las dos listas de forma que consigamos el siguiente *output*. Almacenad los resultados en una nueva variable y llamadla **lista_lista**.

```
lista_lista = [[44, 55, 67, 44, 98, 29], [34, 56, 56, 78, 67, 56]]
```

- Unid las dos listas de forma que consigamos el siguiente *output*. Almacenad los resultados en una nueva variable y llamadla **lista_numeros**.

```
lista_numeros = [44, 55, 67, 44, 98, 29, 34, 56, 56, 78, 67, 56]
```

- De la lista **lista_lista**, ¿cuántas veces aparece el número 67? ¿y en la lista **lista_numeros**. Justificad por qué salen números diferentes
- Usando la **lista_numeros**, dadle la vuelta a la lista.

■ Happy coding ■

Listas 2

Juntar listas predefinidas. Añadir información a una lista. Sacar ciertos datos de la lista. Practicamos con los índices de la lista.

Hasta ahora, las listas que hemos visto las definimos una vez, y luego las modificamos enteras. Sin embargo, una lista es un tipo de contenedor *indexado*. ¿Esto qué significa? Pues como ya vimos usando los métodos `.reverse()` y `.sort()`, el orden de la lista importa y puede ser modificado. En los contenedores *no indexados* el orden es aleatorio, y por tanto no se puede modificar. Es como si ese tipo de contenedores *no indexados* son como bolsas de contenidos, mientras los tipos *indexados* tienen una estructura, parecido a una tabla.

Creamos una lista con las temperaturas de diciembre 2021:

```
temperaturas_diciembre = [10, 10, 13, 14, 12, 15, 12, 11, 13, 17, 18, 15, 15, 17, 17, 13, 13,
```

Como la lista está estructurada, cada contenido tiene un *index*:

temperaturas_diciembre	10	10	13	...	14	17	18
index	0	1	2	...	28	29	30

El índice o *index* en Python empieza en `0`, es decir el primer sitio en la lista tiene el *index* `0`. Nota que diciembre tiene 31 días y que por lo tanto la lista de temperaturas en diciembre contiene 31 temperaturas. Por empezar contando las posiciones en `0`, los índices existen hasta el índice `30` sólo. El índice `31` **no existe**, aunque la lista tiene 31 contenidos.

Otra cosa que destacar antes de empezar con el Notebook es que en Python cualquier input (código) puede pasarse a otra línea sin tener que especificar nada. La indentación de la siguiente línea puede alinearse al contenido que estabas especificando, aunque no es obligatorio.

Recomendaciones a la hora de programar

Abajo tenéis tres ejemplos, que empiezan en las líneas `2`, `9`, y `15` del bloque de código, y que producen exactamente el mismo resultado que el bloque de arriba. Se recomienda usar la primera versión para que tu código sea más fácil de leer. Si quieras aclarar algo en tu código, siempre puedes añadir comentarios con `#` que no serán ejecutado. La recomendación es nombrar las variables para que sea legible el código (es decir, evitar `variable1`, `texto`, `test`, etc.) y así reducir la necesidad de poner comentarios. Ojo que reducir los comentarios no es lo mismo que evitarlos a todo coste.

```
# Se recomienda alinear así:  
temperaturas_diciembre = [10, 10, 13, 14, 12, 15,  
                           12, 11, 13, 17, 18, 15,  
  
                           15, 17, 17, 13, 13, 11,  
                           12, 10, 12, 14, 12, 12,  
                           11, 12, 16, 16, 14, 17, 18]  
  
# Alternativas que funcionan pero que no se recomienda:  
temperaturas_diciembre = [10, 10, 13, 14, 12, 15,  
                           12, 11, 13, 17, 18, 15,  
                           15, 17, 17, 13, 13, 11,  
                           12, 10, 12, 14, 12, 12,  
                           11, 12, 16, 16, 14, 17, 18]  
  
temperaturas_diciembre = [10, 10, 13, 14, 12, 15,  
                           12, 11, 13, 17, 18, 15,  
                           15, 17, 17, 13, 13, 11,  
                           12, 10, 12, 14, 12, 12,  
                           11, 12, 16, 16, 14, 17, 18]
```

Jupyter Notebook



modulo-1-leccion-03-listas2.ipynb 54KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso

- Las listas son mutables
- **LOS ÍNDICES EMPIEZAN EN 0**
- Para acceder a los índices usamos los corchetes

```
lista[indice]
```

- El índice máximo de una lista es

```
lista[len(lista) - 1]
```

que es lo mismo que:

```
lista[-1]
```

- Si queremos acceder a los elementos de la lista empezando por detrás, **empezamos por -1**
- Para acceder al índice de un elemento de la lista usamos el método .index()

```
lista.index(numero_bucamos)
```

- Si queremos acceder a un rango de elementos:

```
lista[:2] # nos devuelve los dos primeros elementos. NO INCLUYE EL ÚLTIMO ELEMENTO
```

```
lista[2:] # nos devuelve del elemento 2 en adelante. INCLUYENDO EL ELEMENTO 2
```

- Podemos saltar elementos: `start:stop:step`

- Métodos de listas:

- `copy()` : hacer una copia exacta de la lista
- `sorted()` : a diferencia del `sort()` no nos aplica los cambios sobre la lista. Los resultados tendremos que almacenarlos en una variable nueva. Pero, de la misma forma que el `sort()` tiene el parámetro `reverse()`
- `append()` : añade elementos a la list

```
lista1 = [2,3,4,5]
lista2 = [7,8,9]
```

```
lista2.append(lista1)
```

```
# output, NOS DEVUELVE UNA LISTA DE LISTAS
[7,8,9, [2,3,4,5]]
```

- `extend()` :

```
lista1 = [2,3,4,5]
lista2 = [7,8,9]
```

```
lista2.append(lista1)
```

```
# output, NOS DEVUELVE UNA ÚNICA LISTA CON TODOS LOS ELEMENTOS
[7,8,9, 2,3,4,5]
```

- `insert()` : inserta un valor en un índice especificado

```
lista3 = [2,3,4,5]
```

```
lista3.insert(3, "Hola")  
  
# output, NOS EL ELEMNTO "HOLA" EN EL INDICE 3  
[2,3,4,"Hola", 5]
```

- `pop()` : nos elimina un elemento. Hay dos formas de hacerlo:

```
lista = [1, 0,2,3]  
  
# si no especificamos nada, nos elimina el último elemento  
lista.pop()  
  
lista.pop(1) # nos elimina el elemento que está en el índice que le especificamos. En este caso, el índice 1
```

- `remove()` : nos elimina elementos de la lista. En este caso especificamos el elemento que queremos eliminar

```
lista = ["Hola", "Que tal", "estamos aprendiendo"]  
  
lista.remove("Hola") # nos eliminará el elemento "Hola" de la lista  
  
# output  
lista = [ "Que tal", "estamos aprendiendo"]
```

Pair programming

En el *pair programming* de hoy seguiremos poniendo en práctica las listas de Python. En la sesión de hoy tendréis que realizar las siguientes tareas:

- Cread una lista con 10 números aleatorios entre el 0 y el 100. Para esto tendréis que ejecutar el siguiente código:

```
import random
lista = random.sample(range(1,100), 10)
```

Este código no tenemos por qué entenderlo. En este código veremos un par de cosas que no sabemos que son:

- `random.sample()` : os dejamos por [aquí](#) la documentación de la `random.sample` por si queréis curiosear.
- `range()` : [aquí](#) tenéis la documentación para las más atrevidas.

Aunque no es obligatorio que investiguéis en profundidad estos links, si que es recomendable. Como futuras analistas y programadoras debemos acostumbrarnos a leer documentación.

- Con la lista creada en el ejercicio anterior:

Nota: cuando os preguntamos el elemento que está en la tercera posición nos referimos a la posición "humana", es decir, dada la siguiente lista:

```
ejemplo = [4,6,2,7,9,0]

# si os pedimos el número que está en tercera posición el resultado deberá ser "2"

# si os pedimos el número que está en primera posición deberá ser "4"
```

- ¿Cuál es la longitud de la lista? Utilizad métodos de Python para extraer este resultado.
- Extraed el número que está en la tercera posición. Recordad que los índices en Python empiezan en 0 y que no corresponden con las "posiciones humanas"
- Extraed el número que está en la tercera posición utilizando los índices negativos.
- Preguntad al usuario por un número de la lista que hemos creado en el primer paso, recordad que para eso tendréis que usar la función `input`. ¿En qué posición está el número que ha elegido la usuaria?
- Extraed los valores que estén entre la tercera posición y la quinta (ambos inclusive). Esto nos tendrá que devolver una lista de 3 elementos.
- Extraed los valores que estén entre las posiciones 6 y 9 utilizando índices negativos. Os debe devolver una lista con 4 números.
- ¿Cuáles son los números que están en las posiciones pares nuestra lista?
- ¿Y los que están en posiciones impares?

- Dadas las siguientes listas:

```
nombres = ["Laura", "Lorena", "Lupe", "Loreto", "Lucía"]

notas = [9, 8.5, 9.8, 8.9, 9.2]
```

- ¿Qué nota ha sacado Lupe? Para este ejercicio os pedimos que uséis primero el método `index()` de listas y luego los índices con `[]` de las listas. Para esto tendréis que sacar primero en qué posición está Lupe en la lista `nombres` y usar el resultado de este paso para acceder, usando los índices de Python, a la nota de Lupe.
- De la misma forma, ¿qué alumna ha sacado la nota más alta? Tendréis que poner en práctica,

métodos aprendidos hoy y en la lección de listas de ayer.

- ¿Qué alumna ha sacado la nota más baja? ¿Es esta nota mayor que 5?
- Ordenad la lista de notas de mayor a menor y almacenalo en una nueva variable llamada `notas_nuevo_orden`
- Ordenad las notas de las alumnas de mayor a menor. Usa el método que sobreescriba la variable `notas`.
- Ordenad la lista de nombres de mayor a menor. ¿Qué es lo que ha pasado?
- Borrard la lista de notas.

- Dadas las siguientes listas:

```
lunes = [34, 56, 82]
martes = [99, 64, 24]
miercoles = [12, 59, 71]
```

- Unid las tres listas de forma que tengamos el siguiente *output*
`[34, 56, 82, [99, 64, 24], [12, 59, 71]]`
- Definid de nuevo las tres listas. Ahora unid las tres listas de tal forma que obtengamos el siguiente *output*:
`[34, 56, 82, 99, 64, 24, 12, 59, 71]`
- Definid de nuevo las tres listas. Insertad los siguientes valores:
 - En la lista lunes un 100 en la primera posición
 - En la lista martes un 100 en la tercera posición
 - En la lista miercoles un 100 en la última posición

■ Happy coding ■

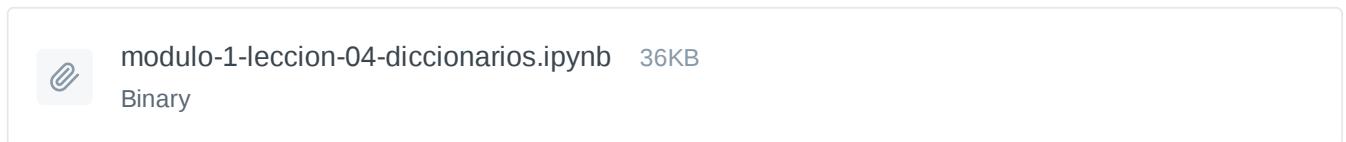
Diccionarios

Alternativa forma de juntar datos, en que cada dato tiene una clave que lo identifique. Manipular diccionarios predefinidos, y modificar contenidos de un diccionario.

Diccionarios

En las listas se puede juntar cualquier cosa, incluso múltiples instantes de lo mismo. La forma de identificar cada elemento de una lista es por índices. En esta clase vamos a mirar otro tipo de contenedor: el diccionario (ingles: dictionary). La diferencia principal es que cada elemento está vinculado a una clave, y que cada clave es única. La forma de identificar cada elemento entonces es por clave. Muchas funciones y métodos te soñarán: los tipos de contenedores tienen bastante en común.

Jupyter Notebook



Reparo

- Son mutables estructuras organizadas de contenedores de datos

- Son tablas, estructuras organizadas de contenidos de datos.

- Se definen con:

```
diccionario = {"key1": [value],  
               "key2": [value] }
```

- Pueden contener cualquier tipo de dato. ESTO PARA LAS KEYS
- Los *values* pueden contener cualquier tipo de TIPO DE DATO O ESTRUCTURA DE DATOS.
- Las *keys* son únicas, no puede haber dos iguales.

- Propiedades:

- `.keys()` : nos devuelve las *keys* del diccionario
- `.values()` : nos devuelve los *values* del diccionario
- `.items()` : nos devuelve una lista de tuplas con todas las parejas *key-value* que hay en el diccionario
- `len()` : nos devuelve el número de parejas *key-value* que hay en el diccionario

- Métodos:

- `.get()` : que nos permite acceder al *value* de una *key* especificada.
- `diccionario["key"]` : es exactamente lo mismo que lo anterior.
Si hacemos lo siguiente podemos cambiar el valor del *value*

```
diccionario["key"] = nuevo_valor
```

- `.setdefault()` : nos busca una *key* dada y si no la encuentra la crea.
- `.update()` : nos actualiza el diccionario. Pudiendo añadir nuevas *keys* o cambiando los valores que ya tenemos.
- `.clear()` : nos vacia el diccionario
- `.pop()` : elimina una pareja *key-value*, de una *key* que nosotras especifiquemos
- `.popitem()` : nos elimina la última pareja *key-value* del diccionario. NO LE PODEMOS PASAR UNA KEY ESPECIFICA
- `.copy()` : nos hacemos una copia exacta del diccionario. NO PODEMOS PASAR UNA KEY ESPECIFICA
- `sorted()` : ordena por defecto las *keys* del diccionario de menor a mayor o en orden alfabético.
 - Si quisieramos ordenar los *values* :

```
sorted(diccionario.values())
```

Pair programming

En el *pair programming* de hoy trabajaremos con diccionarios. El objetivo es poner en práctica todos los métodos aprendidos hasta el momento. Para ello lo primero queharemos será contruir un diccionario paso a paso. Nuestro diccionario estará compuesto por:

- Cuatro *keys* que serán:

- nombres
- apellidos
- edades
- hijos

Cada una de las *keys* tendrá un *value* que será una lista que iremos construyendo con *inputs*. Para crear los *values* seguid los pasos que os indicamos a continuación.

- Cread un *input* que pregunte a la usuaria por tres nombres. Almacenad el resultado en una lista. Esta lista será el *value* de la key `nombres`
- Cread un *input* que pregunte a la usuaria por tres apellidos. Convertir el resultado en lista y este será el *value* de la key `apellidos`.
- Cread un *input* que pregunte a la usuaria por tres edades. Almancenad los resultados en una lista. Esta lista será el *value* de la key `edades`.
- Cread un *input* que pregunte a la usuaria por el número de hijos. Almacenad este resultado en una lista. Esta lista será el *value* de la key `hijos`.
- Cread un diccionario con los resultados del *input*. Al final de todos estos pasos deberíamos tener un diccionario similar a este:

```
{'nombres': ['Lara', ' Alejandro', ' Ana'],
 'apellidos': ['Lopez', ' Martin', ' Solo'],
 'edades': [34, 20, 28],
 'hijos': [2, 0, 1]}
```

- Una vez tengáis creado el diccionario, añadid una nueva *key* donde tengamos la información sobre el número de hermanos.
- ¿Cúales son las *keys* de nuestro diccionario? ¿Y los *values*?
- Para extraer los *values* de una *key* podemos hacerlo de varias formas.
 - Extraed los valores de la *key* `nombres`. Utilizad dos métodos diferentes.
- Extraed los valores de la *key* `nacionalidad`. En caso de que no exista, cread esa *key* con el value "desconocida".
- Ordena los elementos del diccionario basandonos en las *keys* y los *values*. Esto nos devuelve una lista de tuplas(todavía no hemos visto este tipo de datos, pero lo veremos en la lección de mañana). Convertid esta lista de tuplas en diccionario.
- Eliminad la *key* de "nacionalidad"
- Utilizando métodos de Python. ¿Qué persona tiene mayor numero de hijos? ¿Y quién es la más joven?

Tuplas y sets

Dos otros tipos de contenedores con más restricciones. Para qué sirven y cómo usarlos.

Tuplas

Lo primero que tienes que saber de tuplas es que una vez creada una tupla ya no puede cambiar. Son indexables, como listas y diccionarios, pero su orden no cambiará. Cada vez que añadimos o quitamos variables en listas y diccionarios, los índices disponibles y por tanto el índice de los contenidos se veía cambiado. En una tupla, el contenido siempre estará en el mismo índice, es decir las tuplas son *inmutables*.

Como vimos al sacar un elemento de un diccionario, la clave y el valor son devueltos *juntos*, en una tupla. No hay límite a la cantidad de información en una tupla. Es como un paquete de información, estructurado por índices en vez de claves. Son útiles a la hora de trasladar datos.

Sets

Los sets son "sacos" desordenados con contenidos únicos, es decir no se puede meter el mismo contenido más que una vez y no puedes saber *dónde* en el set se encuentra un contenido específico. Los sets luego pueden ser unidos con otros sets, mientras el contenido de un set siempre se mantiene: una vez metido un contenido es inmutable. Lo que sí que permite un set es añadir elementos nuevos o borrar elementos existentes.

Para sumar o restar sets entre ellos hay unas cuantas funciones específicas que miramos en esta clase.

Jupyter Notebook



modulo-1-leccion-05-tuplas.ipynb 34KB
Binary

Descarga este Jupyter y ábrelo en VS Code.



modulo-1-leccion-05-sets.ipynb 41KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Tuplas

- Se definen con paréntesis `()` o con el método `tuple()`
- Son inmutables
- Si definimos una variable con un único valor seguido de una coma será una tupla!!!
- Se pueden crear tuplas desde listas, diccionarios o sets.
- Para modificar una tupla podremos:
 - Convertirla en lista, modificarla y volverla a convertir en tupla
 - Sobreescribiendo sus valores.
- Tienen orden, se indexan igual que las listas. Recordamos que se empieza por 0!
- Métodos
 - `.count()` : nos devuelve cuantas veces aparece un elemento especificado.
 - `.index()` : nos devuelve el índice (posición) de un elemento dado.
- Función `zip()` :
 - Nos devuelve una lista de tuplas
 - Necesitamos usar `list()` para convertirlo a "humano"
 - Agrupa elemento a elemento

```

edades = [33,30, 29]
nombres = ["Lorena", "Marta", "Sandra"]

list(zip(edades, nombres))

# Output
[(33, 'Lorena'), (30, 'Marta'), (29, 'Sandra')]

```

Repaso Sets

- Se definen con llaves {} o con el método set()
- No pueden contener elementos repetidos
- No tienen orden, y por lo tanto, no pueden ser indexados.
- NO SE PUEDE TENER UN SET DE LISTAS!!
- Propiedades y métodos:
 - len() : para saber la longitud
 - .copy() : nos hacemos un copia
 - .pop() : NO RECIBE NINGÚN ÍNDICE PORQUE LOS SETS NO TIENEN ORDEN. Por lo tanto, nos elimina un elemento al azar.
 - .remove() : podemos eliminar un elemento en concreto usando el valor de dicho elemento

```
set1 = {"brcoli", "chorizo", "pasta"}  
  
set1.remove("chorizo")  
  
# mi set quedaría  
set1 = {"brcoli", "pasta"}
```

Si intentamos eliminar un elemento que no existe, nos devolverá un error.

- .discard() : similar al remove, pero en este caso, si no encuentra el elemento que estamos intentando eliminar **NO** nos devuelve un error.
- .add() : añade un **único** elemento
- .update() : nos permite insertar más de un elemento.

- Operaciones

- .union() : nos devuelve todos los elementos de los set. Lo une
- .intersection() : nos devuelve los elementos que son comunes a los dos elementos
- .difference() : aquí el orden importa. Nos devuelve los elementos únicos de uno de los sets.

```
set1.difference(set2)  
  
# nos va a devolver los elementos que son únicos del set1  
  
set2.difference(set1)  
  
# nos va a devolver los elementos que son únicos del set2
```

- .symmetric_difference() : nos muestra todos los elementos menos el que tiene el común.
- .issubset() : nos evalua si un set está contenido al completo en otro set.

```
# dados los siguientes sets  
animales = {"gato", "perro", "lagartija", "tortuga"}  
mamiferos = {"gato", "perro"}  
  
# 1 Están todos los elementos de mamiferos contenidos en animales? Que es lo mismo que  
mamiferos.issubset(animales)
```

```
# Output  
True
```

```
# 2 Están todos los elementos de animales contenidos en mamíferos? Que es lo mismo q  
animales.issubset(mamiferos)  
# Output  
False
```

- `.issuperset()` : evalua si un set contiene elementos del otro set.

```
# dados los siguientes sets  
animales = {"gato", "perro", "lagartija", "tortuga"}  
mamiferos = {"gato", "perro"}  
  
# 1 Están todos los elementos de mamíferos contenidos en animales  
animales.issuperset(mamiferos)  
  
# output  
True
```

```
# 2 Están todos los elementos de animales contenidos en mamíferos  
mamiferos.issuperset(animales)  
  
# output  
False
```

- `.isdisjoint()` : evalua si **TODOS** los elementos de los set son diferentes o no. En este caso el orden **NO** importa

```
# dados los siguientes sets  
animales = {"gato", "perro", "lagartija", "tortuga"}  
mamiferos = {"gato", "perro"}  
  
# son todos los elementos de mamíferos distintos de los animales?  
mamiferos.isdisjoint(animales)  
  
# Output. Es así porque comparten dos elementos en común, "gato" y "perro"  
False.
```

```
# ahora bien, si nuestros sets fueran los siguientes:
```

```
animales1 = {"rana", "raton", "serpiente"}  
peces = {"tiburon", "pez payaso"}  
  
# son todos los elementos de mamíferos distintos de los animales?  
animales1.isdisjoint(peces)  
  
# Output, esto es así porque los dos sets no comparten NINGÚN ELEMENTO.  
True
```


Pair programming

En el *pair* de hoy pondremos en práctica algunos de los conceptos aprendidos de sets y tuplas.
Pongámonos manos a la obra:

Tuplas

- 1- Cread una tupla que contenga la letra "A"
- 2- Añadid los siguientes elementos "d", "a", "l", "a", "b" a la tupla definida en el paso anterior. ¿Se puede? ¿Por qué?
- 3- Pensad de que forma podríamos tener una tupla que contenga el siguiente contenido ("A", "d", "a", "l", "a", "b") usando métodos de Python.
- 4- ¿Cuál es el índice de la letra "l"?
- 5- ¿Cuántas veces aparece la letra "a" en nuestra tupla?

Sets

- 1- Dados los siguientes sets:

```
set1 = {1,3,6,2,8,9}  
set2 = {2,1,9,3,10,18}
```

- 2- Identifica los elementos que están en `set1` pero no en `set2`. Almacenad los resultados en una variable que se llame `set3`
- 3- Identifica los elementos que están en `set2` pero no en `set1`. Almacenad los resultados en una variable que se llame `set4`.
- 4- Identifica los elementos comunes entre el `set1` y el `set2`. Almacenad los resultados en una variable que se llame `set5`.
- 5- Cread un set vacío llamado `set6`. Añadid el `set3` y `set4` al `set6`.
- 6- ¿Es el `set1` igual al `set6`?
- 7- Chequead si el `set1` contiene al `set2`. **Pista:** deberéis usar el método `issubset()`
- 8- Eliminad el primer elemento del `set6`

■ Happy coding ■

Sentencias de control

Entender sentencias booleanas y usarlas en bucles if, y while. Introducimos el bucle try.

Loops 1

En esta lección aprenderemos que es el control de flujo y algunas de las estructuras que podemos usar en Python para poder hacerlo.

Entre ellas podemos encontrar :

Pyhton tiene 3 estructuras de control de flujo:

- Condicionales:

- `if`
 - `if ... else ...`
 - `if ... elif ... else`

- Loops:

- `for`
 - `while`

- Control de los loops con `break` y `continue`

- `break`
 - `continue`

En esta lección aprendemos que son los condicionales y los bucles `while`.

`if ... elif ... else`

Las estructuras de control condicionales, son aquellas que nos permiten evaluar si una o más condiciones se cumplen, para decir qué acción vamos a ejecutar. La evaluación de condiciones, solo puede arrojar 1 de 2 resultados: verdadero o falso (*True* o *False*).

`while`

En Python, los bucles `while` se utilizan para iterar hasta que se cumpla una determinada condición. Básicamente, el bucle se ejecuta tantas veces como la condición siga siendo True. Cuando la condición se convierte en False, el bucle se detiene.

Jupyter Notebook



modulo-1-leccion-06-boolean-statements.ipynb 245KB

Binary

Repaso Sentencias de control

if

- Establecen una condición para que se ejecute el código que esta debajo del `if`. Este código tiene que estar indentado
- La condición nos debe devolver un booleano (`True` o `False`). Los operadores que podemos usar son:
 - `==`
 - `!=`
 - `>`
 - `<`
 - `>=`
 - `<=`
 - `in`
 - `not in`

- Podemos juntar distintas condiciones. Los operadores que usaremos son:

- `and` : se tienen que cumplir **todas** las condiciones

```

numero1 = 3
numero2 = 5

# cuando usamos el operador and se tiene que cumplir las dos condiciones. En este caso
if numero1 > 1 and numero2 > 10:
    print(numero1 + numero2)
else:
    print(numero1)

# output
3
  
```

- `or` : se tiene que cumplir **al menos una de las condiciones**

```

numero1 = 3
numero2 = 5

# en este caso se cumple la primera condición, pero no la segunda. Peeeeero como estamos
if numero1 > 1 or numero2 > 10:
    print(numero1 + numero2)
else:
    print(numero1)

# output
8
  
```

- Si queremos chequear varias condiciones tenemos que usar el `elif`. Podemos poner tantos `elif` como queramos.
- El `if` o el `if ... elif` pueden ir acompañados del `else`. Esta sentencia nos va a agrupar todas las condiciones que no se han cumplido en el código anterior. NO LLEVA CONDICION!!!!!!
- Podemos meter un `if` dentro de otro `if`
-

- El orden importa. Para diferenciarlo del while, este if se parará cuando la condición es True

while

- El código se parará cuando la condición sea False.
- Pueden ser infinitos, cuidado!
- Se pueden incluir condiciones usando `if ... elif ... else` dentro del while.

Cosas curiosas

En Python cuando tenemos una variable o estructura de datos vacía, nos la va a identificar como False. Viendo algunos ejemplo

```
string = ""

if string: # como el string esta vacío Python lo entiende como False, por lo tanto se ejecuta
    print("saludo")
else:
    print("vacio")

# output
"vacio"

# lo mismo pasa con estructuras de datos vacías (listas, tuplas, sets, diccionarios)

lista = []
if lista:
    print("hola")
else:
    print("adios")
# output
"adios"
```

Pair programming

A lo largo de este *pair programming* os presentamos una serie de ejercicios que los tendréis que resolver usando sentencias `if`, `if ... else`, `if ... elif ... else` o `while`.

1. En la escuela donde trabajamos tienen el siguiente sistema de notas:

- Below 25 - F
- 25 to 45 - E
- 45 to 50 - D
- 50 to 60 - C
- 60 to 80 - B
- Above 80 - A

El objetivo de este ejercicio es que le preguntéis al usuario por una nota (numérica) y nosotros le devolvamos la nota con la letra que le corresponde.

2. Cread tres *inputs* donde a cada uno de ellos le preguntéis su edad, el objetivo del ejercicio es determinar quien es el más viejo y el más joven.
3. Escribid un programa que pregunte el nombre del usuario a través de la función *input()*. Si el nombre es "Bond" haced que imprima "Bienvenido a bordo de 007". En caso contrario haced que imprima "Buenos días NOMBRE". (Reemplace NOMBRE por el nombre del usuario).
4. `arbol` es un diccionario que muestra el número de árboles de países por kilómetro cuadrado para países aleatorios con un número considerable de población. Crea una lista llamada "masarboles" que contenga el nombre de los países con más de 40.000 árboles por kilómetro cuadrado.

```
tree = {"Taiwan": 69593,
        "Japan": 49894,
        "Russia": 41396,
        "Canada": 36388,
        "Bulgaria": 24987}
```

5. Los alumnos de un curso se han dividido en dos grupos A y B de acuerdo al sexo y el nombre. El grupo A esta formado por las mujeres con un nombre anterior a la M y los hombres con un nombre posterior a la N y el grupo B por el resto. Escribid un programa que pregunte al usuario su nombre y sexo, y muestre por pantalla el grupo que le corresponde.

6. Escribid un programa que:

- Dada la variable `z` con valor 0, es decir $z = 0$.
- Mientras que el valor de `z` sea menor que 3:
 - Si el valor de `z` es igual a 0:
 - Printead el valor de `z`
 - Almancenad el valor en una lista auxiliar
 - Sumadle 1 al valor de `z`
 - Si el valor de `z` es igual a 1:
 - Printead el valor de `z`
 - Almancenad el valor en una lista auxiliar
 - Sumadle 1 al valor de `z`
 - En caso de que no se cumplan las condiciones anteriores:
 - Printead el valor de `z`
 - Sumadle 1 al valor de `z`

7. El objetivo de este ejercicio es añadir elementos a una lista utilizando el bucle `while`:
- Cread una lista vacía
 - Cread una variable cuyo valor sea 0
 - Mientras que la longitud de la lista creada en el primer paso sea menor que 4:
 - Apendead el valor de la variable creada en el paso 2
 - Sumad 1 a la variable creada en el paso 2
 - Por último, printead los valores que hemos añadido en la lista vacía (ya no lo estaría)
8. El objetivo de este ejercicio es encontrar la suma de números en una lista usando el bucle `while`:
- Cread una variable cuyo valor sea 0, llamadla "i"
 - Cread una lista que contenga los siguientes números: 23,45,12,10,25
 - Cread una variable cuyo valor sea 0, llamadla "suma"
 - Mientras que el valor de la variable "i" sea menor que la longitud de la lista:
 - Sumad elemento a elemento los valores de la lista. Es decir, el primero con el segundo; el resultado de la suma anterior más el tercer número, etc
 - Sumad 1 a la variable "i" por cada iteración por el bucle `while`
 - El resultado esperado es 115.
9. Escribid un programa para una empresa que tiene salas de juegos para todas las edades y quiere calcular de forma automática el precio que debe cobrar a sus clientes por entrar. El programa debe preguntar al usuario la edad del cliente y mostrar el precio de la entrada. Si el cliente es menor de 4 años puede entrar gratis, si tiene entre 4 y 18 años debe pagar 5€ y si es mayor de 18 años, 10€.
10. La pizzería Bella Napoli ofrece pizzas vegetarianas y no vegetarianas a sus clientes. Los ingredientes para cada tipo de pizza aparecen a continuación.
- Ingredientes vegetarianos: Pimiento y tofu.
 - Ingredientes no vegetarianos: Peperoni, Jamón y Salmón.
- Escribid un programa que pregunte al usuario si quiere una pizza vegetariana o no, y en función de su respuesta le muestre un menú con los ingredientes disponibles para que elija. Solo se puede elegir un ingrediente además de la mozzarella y el tomate que están en todas las pizzas. Al final se debe mostrar por pantalla si la pizza elegida es vegetariana o no y todos los ingredientes que lleva.

Happy coding

Bucles for

Trabajar con bucles for, rangos, y list comprehensions.

Loops 2

Parecido a los bucles `while`, se puede definir un bucle `for` que tenga la cantidad de iteraciones que haya contenidos en un iterable (una lista por ejemplo). Para cada contenido se ejecuta el código que haya especificado en el bucle, y al terminar vuelve al inicio del bucle para empezar de nuevo con el siguiente contenido.

También hay formas de alternar este flujo, saltando algunas iteraciones o abandonar el bucle por completo antes de terminarlo. También miramos rangos en esta clase, para especificar listas de números que luego pueden servir como base de un bucle `for`.

Los bucles for se pueden definir de forma muy compacta, incluso tan compacta que caben dentro de otras funciones, como `print()`.

Jupyter Notebook



modulo-1-leccion-07-bucles-for-range-comprehensions-I.ipynb 54KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



modulo-1-leccion-07-bucles-for-range-comprehensions-II.ipynb 28KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

[Aquí](#) tenéis el link del artículo del `break`, `pass` y `continue`.

Repaso bucles for

Bucles for

- Sirven para iterar por todos los elementos de mi variable. Estas variables tienen que ser iterables, pueden ser
 - listas
 - tuplas
 - diccionarios
 - set
 - strings
- NO PODEMOS INTERAR POR UN NÚMERO
- Los bucles for se pueden combinar con `if ... elif ... else`, while, otro for.
- Siempre tenemos que poner los `:`.
- Iteración en diccionarios:
 - Por defecto nos intera por las keys.

```
for key in lista_compra:
    print(key)

# Output, nos devuelve cada una de las keys del diccionario
Brocoli
Chorizo
Zanahoria
Arroz
Pan
```

- Si queremos iterar por los `values` tendremos que usar el método `values` de diccionarios

```
for values in lista_compra.values():
    print(values)

# Output, nos devuelve cada uno de los valores del diccionario
2.45
45
7
1.2
0.95
```

- Si queremos iterar por `keys` y `values`, usaremos el método `.values()`

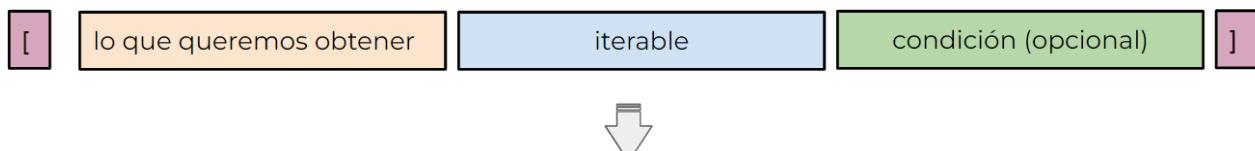
```
for k, v in lista_compra.items(): # nos devuelve tanto las keys como los values Aquí
    print(k, "---", v)

# Output
Brocoli --- 2.45
Chorizo --- 45
Zanahoria --- 7
Arroz --- 1.2
Pan --- 0.95
```

List comprehension

Su principal uso es para comprimir un for loop en una sola línea. Es una estructura de código con la que iremos ganando fluidez con el tiempo. No hace falta que ahora nos sintamos cómodas con ellas o que nos salgan.

- Lo ideal ahora que estamos aprendiendo es, primero hacerlo de forma normal y luego intentar hacerlo con un for loop.
- Su estructura **si queremos hacer un for loop**



Imaginemos que queremos convertir todas las letras a minúsculas



list

try ... except

- Los usaremos para evitar que nuestro código se pare debido a un error en el código.

```
try: # intenta hacer un split del número. Esto nos devolvería un error. Pero como esta dentro
      print(2.split())

except: # después de intentar la línea anterior, si hay error se ejecutará el contenido de
      print("Estas intentando hacer un split de un tipo de un integer")

# Output
"Estas intentando hacer un split de un tipo de un integer"
```

continue, pass y break

- `break` parará la ejecución del bucle.

```
numeros = [1,2,3,4,5,6,7]
```

```
for num in numeros:
    if num == 5:
        break
    else:
        print(num)
# Output
1
2
3
4
```

el código se para en el momento en el que se cumple la condición que pasamos en el `if`.

- `pass` y `continue`: la principal diferencia es que el `*continue*` fuerza a iniciarse el `for`, mientras que el `pass` simplemente obvia la sentencia de código que hay.

```
a = [0, 1, 2]
```

```
# CONTINUE
for element in a:
    if element == 1:
        continue
    print(element) # el print esta fuera del if!!!
```

```
# output. Independientemente de que el código de después del continue este fuera o dentro
0
2
```

PASS, el `pass` solo afectará al código que está en el interior del `for`.

```
for element in a:
    if element == 1:
        pass
    print(element)
# Output, como el pass solo afecta a lo que está en el interior del for, el print que tiene
0
1
2
```

Pair programming

En este ejercicios de *pair programming* podremos en práctica los bucles *for* a través de múltiples ejercicios:

1. Escribid un programa que pregunte al usuario una cantidad a invertir, el interés anual y el número de años, y muestre por pantalla el capital obtenido en la inversión cada año que dura la inversión.
2. Escribid un programa en el que se pregunte al usuario por una frase y una letra, y muestre por pantalla el número de veces que aparece la letra en la frase.
3. Escribid un programa que nos devuelva la tabla de multiplicar de un número especificado en una función *input*
4. Dada la siguiente lista de números:

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

Escribid un programa que muestre sólo los números de una lista que cumplan las siguientes condiciones

- El número debe ser divisible entre cinco
 - Si el número es mayor que 150, se salta y se pasa al siguiente número
 - Si el número es mayor que 500, entonces detenga el bucle
5. Dada la siguiente lista:

```
list1 = [10, 20, 30, 40, 50]
```

Imprimir la lista en orden inverso

6. Dado el siguiente código

```
for i in ['1','2','3']:  
    print (i**2)
```

Usando un `try ... except`, cread un programa para evitar que nos de un error el código anterior. En su lugar, si se encuentra un error, deberéis poner el código necesario para que no nos devuelva error.

Happy coding

Funciones Intro

La definición de funciones propias

Funciones 1

En esta lección aprenderemos que son las funciones, como podemos definirlas, y para que son útiles.

Una función es un bloque de código que nos puede servir en más que una ocasión. En Python tenemos algunas funciones que ya vienen definidas por defecto. Algunas de sus propiedades principales son:

- **Reutilización**: supongamos que escribimos un código que es muy útil para nuestro objetivo, y según avanzamos en nuestro trabajo vemos que lo vamos a necesitar en varias ocasiones. En este caso podemos copiar y pegar una y otra vez el código, pero esto puede resultar algo tedioso si lo tenemos que hacer muchas veces.
Imaginemos ahora que hemos encontrado un error en nuestro código, tendríamos que ir uno a uno cambiándolo, lo que podría llevar a que se nos olvidará en alguna de las copias o nos equivoquemos. La solución para esto, crear una función que realice la tarea que queremos repetir.
- **Modularidad**: las funciones permiten dividir procesos en pasos más pequeños.
Imaginemos que tenemos un código que lee un archivo, procesa el contenido y escribe un archivo de salida. Nosotros nos podremos crear tres funciones, una para cada tarea que sea universal para este archivo pero también para cualquier otro.

Jupyter Notebook



modulo-1-leccion-08-funciones-1-Intro.ipynb 45KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Funciones I

- Se definen usando la palabra clave `def`
- Bloques de código que podemos usar a lo largo del jupyter. Nos ayudan a reutilizar código
- Podemos tener funciones:

- Sin parametros

```
def mi_funcioncita():
    # pasan cosas
    return
```

- Con parámetros: podemos poner todos los parámetros que queramos!

```
def mi_funcioncita(a, diccionario):
    # pasan cosas

    return
```

- Diferencia entre `return` y `print`. **Una función sin `return` nos devuelve `None`**
 - El `print` solo nos printea por pantalla el mensaje que le pasemos. Nos puede resultar util para:
 - Detectar donde nos estamos equivocando en nuestro código
 - Saber en que punto de la interación estamos.

```
# ejemplo
def impares (lista):

    lista_impares = []

    for i in lista:
        try:
            if i % 2 != 0:
                lista_impares.append(i)
        except:
            print("estas teniendo problemas con el elemento", i)
    return lista_impares
```

- **Atención!!!** En una función, en el momento en el que se encuentre un `return` el código se para!!! Ejemplo del kahoot.
- Variables locales vs globales
 - Locales: definidas dentro de una función, no van a ser reconocidas fuera de la función.
 - Globales: son las que hemos visto hasta ahora, que las podemos usar a lo largo de todo el jupyter.

Pair programming

Llegó el momento de poner en práctica las funciones ! Os planteamos 5 ejercicios que tenéis que solucionar usando funciones. Tendréis que usar las estructuras de *control flow* (if ... elif ... else) y *loops* (for, while, etc.), métodos de listas, diccionarios etc.

En cada ejercicio, os dejamos una serie de argumentos con los que tenéis que probar que vuestra función sea capaz de resolverlos. Es decir, vuestras funciones tienen que funcionar para cada uno de los argumentos que os hemos puesto.

Manos a la obra!

Como consejo, si alguno de los ejercicios no os sale, nos os preocupeis, estamos aprendiendo a ser programadoras. En ese caso, dejad la función un rato y seguid con la siguiente!

Happy coding!

1. Cread una función que reciba 2 números enteros en forma de *string* como entrada, y dé como resultado la suma (también en forma de *string*)

Condiciones:

- Si los dos parámetros que recibe la función son *strings* vacíos la función nos debe devolver "0"
- Si el primer parámetro es un *string* vacío, la función nos devuelve el valor del segundo parámetro y viceversa.

Probad la función para las siguientes combinaciones de "números"

```
"4", "5" --> "9"  
"34", "5" --> "39"  
"", "" --> "0"  
"2", "" --> "2"  
"-5", "3" --> "-2"
```

2. El objetivo es comparar cada par de enteros de 2 listas, y devolver una nueva lista con el número mayor de la comparación..

Probad la función para las siguientes listas:

```
arr1 = [13, 64, 15, 17, 88]  
arr2 = [23, 14, 53, 17, 80]  
resultado = [23, 64, 53, 17, 88]
```

Pista En Python existe el método [zip](#).

3. El objetivo de este ejercicio es convertir un *string* en un nuevo *string* en el que cada carácter del nuevo *string* es "(" si ese carácter aparece sólo una vez en el *string* original, o ")" si ese carácter aparece más de una vez en el *string*. Nuestro código **no** tiene que ser case sensitive, es decir, si hay una letra en mayúscula y en minúscula cuenta como dos apariciones.

Probad la función para los siguientes strings

```
"din"      => "((("  
"recede"   => "()()()"  
"Success"  => ")())()"  
"(( @"     => "))(( "  
"Ocvl@GamFLAFkixkS" => "((((((())))))))((("
```

BONUS 4. A Pete le gusta hacer pasteles. Tiene algunas recetas e ingredientes. Desgraciadamente, no se le dan bien las matemáticas. ¿Puedes ayudarle a averiguar cuántas tartas puede hacer teniendo en cuenta sus recetas?

Escribid una función, que tome la receta (diccionario) y los ingredientes disponibles (también diccionario).

```

Probad con los siguientes diccionarios:

```
RECETA {'flour': 500, 'sugar': 200, 'eggs': 1},
INGREDIENTES {'flour': 1200, 'sugar': 1200, 'eggs': 5, 'milk': 200}
```

RESULTADO: 2

-----

```
RECETA {'apples': 3, 'flour': 300, 'sugar': 150, 'milk': 100, 'oil': 100}
INGREDIENTES {'sugar': 500, 'flour': 2000, 'milk': 2000}
```

RESULTADO 0

```

****Pista**:** Nuestra función recibirá dos parámetros.

BONUS 5. Escribid una función que va a ser una calculadora. La lista recibirá dos parámetros, una lista de números y un *string* con la operación que queremos hacer (puede ser "*", "+")

```

Probad con los siguientes inputs

```
lista1 = [11, 6, 98, 1, 2] , "*"
RESULTADO 12936
```

```

lista2 = [23,34, 56, 11, 90] , "+"
RESULTADO 214
```

```

lista3 = [23, 4,109 , 94, 77] , "-"
RESULTADO 'Lo siento, necesito que me pases una operación valida.'
```
```

Happy coding

Funciones Argumentos

En esta lección aprendemos algunos conceptos más avanzados sobre funciones, como los argumentos por defecto, los args, kwargs y la recursividad

Funciones 2

En la clase anterior, aprendimos algunos conocimientos básicos de las funciones de Python. Hoy seguiremos aprendiendo nuevos conceptos sobre ellas. Estos conceptos incluyen:

- Argumentos por defecto: indican que el argumento de la función tomará ese valor si no se pasa otro valor cuando llamaremos a la función.
- Args: Es una **lista** de argumentos, como argumentos posicionales. Usaremos este tipo de argumentos lo usaremos con listas o tuplas, y cuando no sepamos el número de parámetros que va recibirán nuestra función.
- Kwargs : de la misma forma que los args usaremos este tipo de estructura cuando no sepamos el número de elementos que recibe nuestra función. Serán **diccionarios** cuyas keys se convierten en parámetros y sus valores en los argumentos de los parámetros.
- Funciones recursivas: son funciones que se llaman a sí mismas.

Jupyter Notebook



modulo-1-leccion-09-funciones-2-argumentos.ipynb 32KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Funciones II

Parametros por defecto:

- Son parámetros que vamos a definir su valor cuando creamos la función.
- Se pueden definir los que queramos
- Siempre van al final, y dentro de los paréntesis. INCLUSO CUANDO TENGAMOS ARGS Y KWARGS
- Pueden ser números, strings, listas, diccionarios, etc.
- Aunque estén definidos. Podemos cambiar su valor cuando llamemos a la función

Args:

- Los usaremos cuando desconocemos el número de argumentos que le vamos a pasar a nuestra

- Los usaremos cuando desconocemos el número de argumentos que le vamos a pasar a nuestra función

- El tipo de dato que debemos incluir son listas y tuplas.
- Para definirlos en nuestra función hay que usar `*`
- Si le pasamos una lista cuando llamamos a la función hay que poner `*`.
- Cada elemento de la lista es un parámetro. Si le pasamos una lista de listas, cada sublista será un parámetro de mi función.
- Los podemos pasar de dos formar diferentes:

```
# definimos la función
def sumar_numeros(*args):
    return sum(args)

# definiendo una lisra previamente
numeros_lista = [1,2,3,4]

# y cuando llamemos a la función le añadimos un *
sumar_numeros(*numeros_lista)

# o pasando los elementos por separado
sumar_numeros(1,2,3,4)

# Las dos formas nos devuelven exactamente lo mismo
```

Kwargs:

- Son diccionarios. Es un diccionario con todos los pares `key : value` que queramos.
- Las `keys` son los parámetros y los `values` son los valores que tomaran esos parámetros dentro de la función.
- Le tendremos que poner `**` cuando los definamos en la función y cuando llamemos a la función.
- El orden importa, cuando trabajamos con args y kwargs el orden es ARGS y KWARGS.

Recursividad:

- Son funciones que se llaman a si mismas
- Para parar estas funciones, necesitamos poner una condicion de parada, ya sea un condición con un if o con un while o incluso con un for.

Pair programming

Seguimos trabajando con funciones, en esta sesión seguiremos la misma dinámica que en la sesión anterior.

Manos a la obra!

Como ayer, si os alguna de las funciones no os sale, no os preocupeis! Seguid trabajando en otros ejercicios.

Happy coding!

1. Vamos a crear una "Calculadora de puntos". Tenéis que escribir una calculadora que reciba cadenas de caracteres como entrada. Los puntos representarán el número de la ecuación. Habrá puntos en un lado, un operador, y puntos de nuevo después del operador. Los puntos y el operador estarán separados por un espacio.

Aquí os dejamos los operadores válidos:

- Suma
- Resta
- Multiplicación
- División entera

Vuestro trabajo

Tendréis que devolver un *string* que contenga puntos, tantos como devuelva la ecuación. Si el resultado es 0, devuelve la cadena vacía. Cuando se trata de una resta, el primer número siempre será mayor o igual que el segundo.

Probad la función con los siguientes casos:

```
"..... + ....." => "....."
"..... - ..." => "..."
"..... * ..." => "....."
"..... // ..." => "..."
". // .." => ""
".. - .." => ""
```

2. Te despides de tu mejor amigo, "Nos vemos el próximo año".

Vuestro trabajo: Dado un año, encuentra el próximo cumpleaños o el año más cercano en el que verás a tu mejor amigo.

Condiciones

- Año siempre positivo.
- El siguiente año que le felicites a tu mejor amigo no puede tener ningún dígito repetido.

Probad la función con los siguientes casos:

7712 ==> El siguiente año que felicitarás a tu amigo será el 7801.
Por que es el siguiente año en el que no hay ningúun dígito repetido.

1001 => 1023

1123 => 1203

2001 => 2013

3. Tenéis que crear un función que chequee la vida de un evaporador que contiene un gas.

Conocemos el contenido del evaporador (contenido en ml), el porcentaje de gas que se pierde cada día y el umbral en porcentaje a partir del cual el evaporador deja de ser útil. Todos los números serán estrictamente positivos.

⚠ Nota: el contenido no es, de hecho, necesario en el cuerpo de la función, podéis utilizarlo o no.

Probad con los siguientes casos:

```
10, 10, 5 => 29
```

```
10, 10, 10 = > 22
```

4. Definid una función que tome como argumento un entero y devuelva True o False dependiendo de si el número es primo o no.

Según la Wikipedia, un número primo es un número natural mayor que 1 que no tiene divisores positivos más que 1 y él mismo.

Probad la función con los siguientes números:

```
0 => False  
2 => True  
73 => True  
-1 => False  
5099 => True
```

5. Probablemente conozcais el sistema de "me gusta" de Facebook y otras páginas. La gente puede dar "me gusta" a las publicaciones del *blog*, a las imágenes o a otros elementos. Queremos crear el texto que debe mostrarse junto a dicho elemento.

Cread una función que toma una lista que contiene los nombres de las personas a las que les gusta un artículo. Debe devolver el texto que se muestra en los ejemplos:

Probad los siguientes ejemplos:

```
[] --> "A nadie le gusta esto"  
["Paola"] --> "A Peter le gusta esto"  
["Jacoba", "Alex"] --> "A Jacob y Alex les gusta esto"  
["Maria", "Juana", "Lola"] --> "A Max, John y Mark les gusta esto"  
["Alex", "Jacoba", "Lola", "Carmen"] --> "A Alex, Jacob y 2 más les gusta esto"  
["Alex", "Jacoba", "Lola", "Carmen", "Mariana"] --> "A Alex, Jacoba y 3 más les gusta esto"
```

Nota: Para 4 o más nombres, el número en "y otros 2" simplemente aumenta.

Clases

en esta clase aprenderemos que son las clases y como podemos crearlas en Python

Clases

Las clases proporcionan un medio para estructurar nuestro código de manera que las propiedades y los comportamientos se agrupen en objetos individuales.

Dicho de otro modo, las clases son un enfoque para modelar cosas concretas del mundo real, como las frutas, los coches etc. La programación orientada a objetos modela las entidades del mundo real como objetos de software que tienen algunos datos asociados y pueden realizar ciertas funciones.

Aunque lo tenéis en el jupyter, aquí os dejamos algunos de los conceptos más básicos para empezar con las clases

- **Clase (Class)** : una clase es un esquema del objeto. Es un molde sobre el que crearemos los objetos o instancias con las mismas características (color, forma, sabor, etc.) Objeto o Instancia: es la entidad individual.
Es una fruta concreta, la mandarina que es de color naranja, redonda, con sabor dulce/ácido etc.
- **Atributos** : son las características que le damos a un objeto dado. En nuestro ejemplo:
 - color: naranja
 - forma: redonda
 - sabor: dulce/ácido
- **Métodos** : son las funciones que definen cada objeto.

Jupyter Notebook



modulo-1-leccion-10-Clases-new.ipynb 39KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso de Clases

- Los nombres de las clases siempre serán con la primera letra en mayúscula

- Los nombres de las clases siempre serán con la primera letra en mayúscula.
- Cuando hagamos cualquier cambio en nuestra clase, tendremos que ejecutar la celda de nuevo e instanciar la clase (es decir, llamarla de nuevo)
- Palabras clave:
 - **Métodos**: cada una de las funciones que tenemos dentro de la clase
 - **Atributos**: las características que hacen único a cada objeto. Por ejemplo: Lucia, sus atributos son pelo rizado, Galicia, blog de cocina. Chloe sus atributos son pelos liso, Jaen, le gustan las hierbas. Son los que definimos en el método constructor.
 - **Objetos**: cada una de las "galletas" que creamos con el molde. En nuestro caso, nuestros objetos son Lucia y Chloe.
- Método constructor:

```
def __init__(self)
```

- Es el primer método que tendremos que poner. PERO OJO! NO ES OBLIGATORIO PONERLO
- Es la sección de la clase donde se definen las características base de la clase. Es decir, **los atributos**, tipo de pelo, donde están nuestras chicas, hobbies.
- Los atributos tienen que ir definidos dentro del método, para eso usaremos el self

```
def __init__(self, tipo_pelo, ciudad, hobbies):
    self.tipo_pelo = tipo_pelo
    self.ciudad = ciudad
    self.hobbies = hobbies
```

Será como el caminito de baldosas amarillas que le vaya indicando a nuestra clase de donde "vienen" los atributos.

- Cuando queramos usar uno de estos atributos fuera del método constructor, deberá ir siempre acompañado del `self`

```
def __init__(self, tipo_pelo, ciudad, hobbies):
    self.tipo_pelo = tipo_pelo
    self.ciudad = ciudad
    self.hobbies = hobbies

def vacaciones(self):
    print(f"En tus vacaciones te gusta hacer {self.hobbies}") # como "hobbies" esta en
```

- Clases hijas:
 - Son una copia de otra clase. Van a heredar todo (constructor y métodos) de la clase madre.
 - Pero podemos cambiar los métodos de la clase madre en la clase hija. Simplemente hay que volver a crear el método dentro de la clase hija

```
class Dog:
    def walk(self):
        return "*walking*"

    def speak(self):
        return "Woof!"

class JackRussellTerrier(Dog): # la clase hija hereda todos los métodos de su madre. Pe
```

```
def speak(self):
    return "Arff!"
```

- Para saber si una clase es hija de otra, usamos `isinstance`

```
isinstance( bobo_junior, Dog )
```

Pair programming

En este ejercicio de *pair programming* crearemos una clase. Imaginemos que nuestro jefe nos pide que creemos una clase para tener más ordenada la información sobre cada uno de los empleados de la empresa. Para ello:

- Definid una clase que se llame `Empleados`
- Definid los siguientes atributos que caracterizan a nuestros empleados:
 - nombre
 - apellido
 - edad
 - posición que ocupa en la empresa
 - año en que entró en la empresa
 - número de días de vacaciones que tiene
 - Las herramientas que usa cada uno de ellos, por ejemplo, Outlook, excel y word. Deberá ser una lista.
- Cread una instancia para la clase creada
- Definid los siguientes métodos:
 - descripción
 - calculo_vacaciones
 - cambiando_posicion

Tips para definir los métodos:

- Método `descripcion`:

- El `return` deberá ser un `string` en el que aparezca el nombre, apellidos, los años que lleva en la empresa y los días de vacaciones que le quedan. Este método deberá devolver algo como esto:

```
"El/ella, es Lorena, Data Analyst, quien lleva con nosotros en la empresa desde 2000. A
```

- Método `calculo_vacaciones`:

- Deberá contener `input` que nos pregunte cuántos días te quieres ir de vacaciones.
- Luego deberemos restar el número de días que se fue de vacaciones a el número total de vacaciones que tiene ese empleado.
- Una condición que chequee si:
 - Si el numero de vacaciones que tiene el empleado es igual a 0, devolved un mensaje que diga que se le acabaron las vacaciones
 - Si el número de días que se quiere ir de vacaciones es mayor que el número de vacaciones que le quedan, devolved un mensaje que diga que no se puede ir tantos de vacaciones
 - En caso de que no se cumplan ninguna de estas condiciones, que nos devuelva el número de vacaciones que le quedan al empleado.

- Método `cambiando_posicion`:

- Iteraremos por la lista de herramientas que usa el empleado:
 - Si el empleado usa "Python" devolveremos un mensaje de bien hecho!
 - Si el empleado usa "Excel" le recomendaremos amigablemente que deje excel y empiece a usar Excel

Happy coding

Gestión de Ficheros

Escribir en archivos de texto, abrir archivos xml

Tratamiento de Ficheros 2

En esta clase ampliamos el tratamiento de archivos de texto a no sólo leerlos sino también manipularlos. Creamos archivos de texto, y miramos la diferencia entre sobre escribir y añadir textos nuevos. También tocamos a archivos xml, qué son, qué estructura tienen, y cómo leer los contenidos.

Hemos estado trabajando en una ubicación específica sin que nos importaba mucho hasta ahora. Sin embargo, si tienes unos datos en un archivo externo querrás saber dónde está, cómo navegar a ello, y cómo leer sus contenidos. Para poder hacerlo necesitamos unas funciones de un *module*, y os explicamos cómo cargar un module entero o solo funciones específicas.

Jupyter Notebook



modulo-1-leccion-11-ficheros.ipynb 80KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Otros archivos



ficheros2.txt 0B
Binary

Descarga este archivo. Lo necesitarás en la clase.



contacto.xml 921B
Binary

Descarga este archivo. Lo necesitarás en la clase.



ejercicio1.xml 724B
Binary

Descarga este archivo. Lo necesitarás en los ejercicios.

Repaso Gestión ficheros

Librería os y gestión ficheros

- Para usarla deberemos importarla

```
import os
```

Como ejercicio de buena práctica, los import siempre irán al inicio del jupyter.

- Cuando importamos librerías podremos usar alias. Nos servirán para:

- Para ser prácticos, y evitar tener que poner el nombre completo de toda la librería

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

- Podemos llamar a una sola función de la librería, usando from.

```
from os import listdir # Solo podremos usar este método de la librería si intentamos usa
```

- Rutas relativas vs rutas absolutas:

- Con las rutas relativas podremos trabajar desde cualquier ordenador.
- Las rutas absolutas, solo nos valdrá para nuestro ordenador.

```
/c/Users/agarc/Documents/Adalab/materiales-da-promo-b/modulo-0
```

- Librería `os` : para poder movernos entre directorios y acceder a carpetas y archivos

- Librería `shutil` : para eliminar carpetas y subcarpetas de forma recursiva

- Librería `parse` : para poder trabajar con xml y poder abrir los ficheros correctamente.

- Librería `os`

- `os.getcwd` : nos devuelve la ruta absoluta de el sitio de trabajo
- `os.listdir` : para saber que archivos y carpetas tenemos en el espacio de trabajo
- `os.mkdir` : crea un directorio nuevo. En caso de que el directorio que creamos exista nos dará error
- `os.rename` : nos permite cambiar el nombre de una carpeta
- `os.remove` : que nos permite eliminar ficheros.
- `os.chdir` : nos permite cambiar la carpeta de trabajo.
- `rmdir` : nos permite eliminar carpetas

- Librería `shutil`:

- `rmtree` : que es el que nos permite eliminar de forma recursiva

- Trabajar con ficheros:

- `open()` y `close()` : nos sirven para abrir y cerrar ficheros desde Python. Cuando tenemos un fichero abierto con `open` NO PODREMOS ELIMINARLO, CAMBIARLE NOMBRE O MOVER UBICACIÓN.

- El método `open` :

- Necesita la ruta para abrir el fichero. Hemos visto que es mejor hacer con una ruta relativa
- `encoding` : para que nos traduzca a humano.
- `mode` : permisos de acción sobre el fichero. Tipos:

- `r` : de lectura.
 - `w` : de escritura.
 - `x` : para crear un fichero nuevo. Si intentamos crear un fichero que ya existe nos dará error.
 - `a` : nos permite añadir contenido al fichero sin borrar lo que teníamos antes.
 - `b` : si queremos traducir de binario a humano
 - `t` : si queremos traducir de texto, viene por defecto.
- Al método `open()` le tenemos que pasar la ruta y el nombre del fichero que queremos abrir.

```
f = open(mi_ruta/nombre_fichero.txt)
```
 - El método `.read()` : nos permite leer un archivo, que previamente hemos abierto con `open()`.

```
f.read()
```
 - Para leer los 5 primero caracteres

```
f.read(5)
```
 - Para leer en vez de caracteres, líneas usamos el `readlines()` . Si queremos leer las 5 primeras líneas

```
f.readlines(5)
```

 - `read` : nos permite leer el contenido del fichero. Le podemos especificar cuántos CARACTERES queremos que nos muestra entre paréntesis.
 - `readline` : lee un línea del fichero, si queremos leer varias, tendremos que ejecutar este comando tantas veces como líneas queramos.
 - `readlines` : lee TODAS las lineas de nuestro archivo. Nos devuelve una lista!! Y podremos acceder a cada una de las líneas usando los índices de listas.
 - `with` lo usaremos para poder abrir un fichero y realizar tareas específicas con él. En este caso necesitaremos un alias (as `lo_que_queramos`).

xml

- Para trabajar con estos ficheros deberemos importar `xml.etree.ElementTree`

```
import xml.etree.ElementTree as ET
```

- Para abrir el fichero usaremos el método `.parse()`

```
fichero_xml = ET.parse('nombre_fichero') # Nota, en caso de que no este en nuestro direc
```

- Para acceder a la raíz del archivo usaremos `getroot()`

```
# siguiendo el ejemplo anterior
```

```
raiz = fichero_xml.getroot()
```

- Para acceder a los hijos y nietos del fichero, usaremos un for loop.
- Para extraer el contenido de una etiqueta usaremos el método `find()`. ESTE SOLO NOS DEVUELVE UNO, el primero
- En caso de que queramos acceder a todos los contenidos de una etiqueta usaremos `findall()`

--->

Pair programming

Sigamos poniendo en práctica los conocimientos aprendidos hasta ahora. Es el momento de ponernos manos a la obra con los ficheros. **NOTA** Estos ejercicios los tendrás que solucionar usando funciones

*Objetivo de este ejercicio de **pair programming**:

Creación de carpetas y movimientos entre ellas

----- PRIMERA FUNCIÓN -----

Esta función debe incluir el código que:

1. Nos muestre en qué carpeta estamos trabajando.
2. Cree una carpeta que se llame "aprendiendo-ficheros". \triangle Tened en cuenta que si la carpeta ya existe no la podemos crear, nos devolverá un error. Incluye en la función un programa que evite que nos de un error si la carpeta ya existe.
3. Cree otra carpeta que se llame "datos" dentro de la carpeta "aprendiendo-ficheros". En esta carpeta "datos" guardaremos el fichero "saludo.txt" que os habéis descargado.
4. Cambiad el directorio de trabajo a la carpeta "datos". Antes de seguir chequead que estáis trabajando en la carpeta "datos".
5. Cambiad el nombre de la carpeta creada en el punto 2 a "primera-toma-contacto"

Pistas para resolver este ejercicio Esta función debe incluir el código que:

1. La función tendrá que recibir 3 parámetros:
 - El nombre del nombre de la primera carpeta
 - El nombre de la carpeta segunda carpeta
 - El nombre con el que queremos cambiar el nombre de la primera carpeta creada
2. Para saber si las carpetas ya existen tendrás que usar `lstdir` (recordad que nos devuelve una lista de ficheros y carpetas).
3. Para poder controlar los errores tendremos que usar un if loop, que si el fichero existe, nos devuelva un mensaje de que el fichero ya existe. En caso de que no exista, los deberás crear y que la función nos muestre un mensaje de que se ha creado.
4. Tendrás que ir cambiando de directorio para poder crear las carpetas y cambiar sus nombres.

----- SEGUNDA FUNCIÓN -----

Antes de empezar, recordad descargaros el fichero `saludo.txt` y guardarla en el repo en el que estáis trabajando, dentro de una carpeta que se llame "datos".

1. Lea el fichero que se llame "saludo.txt y muestre su contenido completo.
2. Muestra la línea 4 del fichero

Pistas para resolver este ejercicio

1. Antes de empezar, tendrás que saber cuál es vuestro directorio de trabajo.
2. Tened en cuenta en qué carpeta estáis. Si vuestro directorio de trabajo no es "datos" tendrás que cambiarlo o poner la ruta relativa a la carpeta "datos".
3. Usar `if ... else` para evitar que se nos pare el código.
4. Para cambiar el fichero podrás usar el comando `input` para preguntar el usuario donde está el fichero y que se pueda usar la ruta relativa o absoluta.



saludo.txt 182B
Binary

Descarga este fichero para hacer el ejercicio.

Lectura y escritura de ficheros

1. La función debe recibir 4 parámetros:
 - Nombre del fichero, incluyendo la extensión.
 - Como queremos leer el fichero.
 - El encoding del fichero.
 - El contenido que queremos escribir en el fichero.
2. Condiciones de la función:
 - Si el fichero no existe, debe crearlo, insertar contenido y mostrar su contenido.
 - Si el fichero existe pregunta al usuario si quiere sobreescibirlo. En caso de Si, sobreescribe el fichero, insertad contenido y leedlo. En caso de No, no hace nada.

Pistas

- Para chequear si un fichero existe, podemos usar la función `os.listdir()` que devuelve una lista con los nombres de los ficheros que hay en un directorio.
- Para crear un fichero, podemos usar la función `open()` que recibe como parámetros el nombre del fichero, el modo de apertura y el encoding.
- Para chequear si el fichero existe debéis usar los for loops y el método `in` para comprobar si el nombre del fichero está en la lista de ficheros.
- Usar un `input` para preguntar al usuario si quiere sobreescribir el fichero.
- Para sobreescribir el fichero, podemos usar la función `open()` que recibe como parámetros el nombre del fichero, el modo de apertura y el encoding.
- Strings que cubren más que una línea tienen que ser iniciados y cerrados con tres comillas `"""`.

El contenido que tenéis que insertar es:

Thu Oct 31 08:11:39 2002
Return-Path: <bensul2004nng@spinfinder.com>
X-Sieve: cmu-sieve 2.0
Return-Path: <bensul2004nng@spinfinder.com>
Message-Id: <200210311310.g9VDANT24674@bloodwork.mr.itd.UM>
From: "Mr. Ben Suleman" <bensul2004nng@spinfinder.com>
Date: Thu, 31 Oct 2002 05:10:00
To: R@M
Subject: URGENT ASSISTANCE /RELATIONSHIP (P)
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
Status: 0

Dear Friend,

I am Mr. Ben Suleman a custom officer and work as Assistant controller of the Customs

After the sudden death of the former Head of state of Nigeria General Sanni Abacha on

I declared only (5) five boxes to the government and withheld one (1) in my custody due to US\$55 Billion Dollars (ref:ngrguardiannews.com) of July 2nd 1999. Even the London content will not be discovered. Now that all is calm, we (myself and two of my colleagues)

However as government officials the Civil Service Code of Conduct does not allow us to earn more than our salary on the average, thus our handicapp and our need for your help.

Therefore we want you to assist us in moving this money out of Nigeria. We shall definitely move the consignment to their affiliate branch office outside Nigeria through diplomatic channels.

This business is 100% risk free for you so please treat this matter with utmost confidence.

Expecting your response urgently.

Best regards,

Mr. Ben Suleman

Wed Oct 30 21:41:56 2002
Return-Path: <james_ngola2002@maktoob.com>
X-Sieve: cmu-sieve 2.0
Return-Path: <james_ngola2002@maktoob.com>
Message-Id: <200210310241.g9V2fNm6028281@cs.CU>
From: "MR. JAMES NGOLA." <james_ngola2002@maktoob.com>
Reply-To: james_ngola2002@maktoob.com
To: webmaster@aclweb.org
Date: Thu, 31 Oct 2002 02:38:20 +0000
Subject: URGENT BUSINESS ASSISTANCE AND PARTNERSHIP
X-Mailer: Microsoft Outlook Express 5.00.2919.6900 DM
MIME-Version: 1.0

Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from quoted-printable to 8bit by sideshowmel.si.UM id g9V2foW24:
Status: 0

FROM:MR. JAMES NGOLA.
CONFIDENTIAL TEL: 233-27-587908.
E-MAIL: (james_ngola2002@maktoob.com).

URGENT BUSINESS ASSISTANCE AND PARTNERSHIP.

DEAR FRIEND,

I AM (DR.) JAMES NGOLA, THE PERSONAL ASSISTANCE TO THE LATE CONGOLESE (PRESIDENT LAU

THE INCIDENT OCCURRED IN OUR PRESENCE WHILE WE WERE HOLDING MEETING WITH HIS EXCELLEN

MY PURPOSE OF WRITING YOU THIS LETTER IS TO SOLICIT FOR YOUR ASSISTANCE AS TO BE A CO

YOU WERE INTRODUCED TO ME BY A RELIABLE FRIEND OF MINE WHO IS A TRAVELLER,AND ALSO A I

THE (USD\$25M) WAS PART OF A PROCEEDS FROM DIAMOND TRADE MEANT FOR THE LATE PRESIDENT I

AS A MATTER OF FACT, WHAT I URGENTLY NEEDED FROM YOU IS YOUR ASSISTANCE IN MOVING THIS

THE REMAINING 5% WILL BE USED TO OFFSET ANY COST INCURRED IN THE CAUSE OF MOVING THE I

FINALLY, IT IS IMPORTANT ALSO THAT I LET YOU UNDERSTAND THAT THERE IS NO RISK INVOLVEI

LOOKING FORWARD TO YOUR URGENT RESPONSE.

YOUR SINCERELY,

MR. JAMES NGOLA.

Archivos XML

Aquí tenéis un archivo llamado `peliculas.xml` que contiene una lista de películas.



`peliculas.xml` 3KB

Binary

Descarga este fichero para hacer el ejercicio.

En este ejercicio tendréis que crear una función que reciba el nombre del archivo xml y que devuelva lo siguiente:

- Qué *tag* y atributos tiene el archivo xml.
- La descripción de cada una de las películas que tenemos en ese archivo.
- Los años en que fueron estrenadas las películas.

Pistas

- La función debe recibir un parámetro, el nombre del archivo xml.
- Tendréis que utilizar un bucle for para recorrer todo el archivo xml y extraer la información que os pedimos.
- Recordad el método `.text` para extraer el texto de un elemento.

Happy coding!

Expresiones regulares

Técnicas para buscar ciertas palabras o caracteres en strings.

Expresiones regulares

Con expresiones regulares nos referimos a un "código" para describir una serie de caracteres. A la hora de tener que limpiar datos nos viene muy bien poder encontrar por ejemplo todos los doble espacios y convertirlos a un sólo espacio, o encontrar en un texto enorme de forma automática todas las veces que escribí MAdrid en vez de Madrid. Otro ejemplo serían las fotos que tienes en el móvil, guardado con la fecha en un formato conocido. Si las quisiera juntar con otras fotos de mi pareja podríamos renombrar los archivos a un formato común. Es decir, convertir 'IMG_2022-2-28' y 'DSC28022022' en '2022-02-28-Maria' y '2022-02-28-Paco'. La *descripción* de los strings que buscamos, y sobretodo las variaciones que permitimos que tenga, suelen resultar en una formula bastante complicada. Sin embargo, si la lees pasito a pasito verás que al final sí que entiendes lo que quiere decir.

Jupyter Notebook



modulo-1-leccion-12-expresiones-regulares.ipynb 58KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



soluciones_regex.txt 424B

Binary

Descarga este archivo para ver las soluciones de los ejercicios de regexone.

Repaso Expresiones regulares

- Modulo re
- import re es necesario para poder utilizar regex en Python
- Con findall buscamos todas las incidencias de un patrón
 - devuelve una lista de trozos del string
- regex es una secuencia de caracteres para formar un patrón de búsqueda
- * para un número indeterminado de caracteres, 0 también vale
- ? para 0 o 1
-
- para 1 o más
 - por defecto buscan lo máximo posible, con la interrogación justo detrás busca el mínimo de repeticiones posibles *?, ??, +?
- \w para words, todos los símbolos que necesito para escribir palabras
 - _ también cuenta para \w
 - \W lo contrario, y por tanto es puntuación
- \s para espacios, tabulaciones nuevas líneas, todo que sea “vacío”
- \d busca dígitos, \D hace lo contrario
- definir sets
 - [A-Z] serían todas las letras en mayúsculas
 - [a-z] todo minúsculas
 - [a-f] sería a b c d e f
 - [^A-F] todo lo que no sea una A B C D E o F
 - [gP#] sería cualquier de estas 3 caracteres
- ^texto\$ significa que empieza con texto y termina con texto no en una sola línea sino en el string entero
- {} es para especificar repeticiones
 - [Aa]{2,3} me devuelve
 - aA sí
 - aAa también
 - A no
 - AAAa tampoco
 - sólo repeticiones de 2 o 3 veces lo que hay delante
- re.findall('patron', stringOG) nos devuelve una lista de los strings que cumplen con el patrón
- re.search('patron', stringOG) nos devuelve un match object con las características del primer elemento que cumple con el patron

Pair programming

En este ejercicio pondremos en práctica los conocimientos aprendidos sobre métodos de *strings* y expresiones regulares.

Antes de empezar:

Cargad el fichero del email que tenemos al final de la explicación del pair de hoy y almacenarlo en una variable. Para abrirlo tendremos que usar el método `open` aprendido en lecciones anteriores.

Importad las librerías necesarias para poder usar las expresiones regulares

Recordad que en regex tenemos que buscar patrones para cada objetivo que os planteamos. Tendremos que poner en práctica todo lo aprendido hasta ahora. For loops, métodos de *strings*, indexación de listas, etc.

Los **objetivos** de la sesión de hoy son:

1. Extraer el email de la persona que envió el email (os deberían salir 2).

Pista Tendremos que hacer dos busquedas:

- Una para extraer toda la información del remitente. El remitente siempre empieza por `From:`

```
info_remitente = re.findall('From:.*',email)
```

- Otra para extraer el email únicamente.

2. Extraer el nombre de la persona que envió el email (os deberían salir 2).

Pista Tendremos que hacer dos busquedas:

- Podemos usar el resultado de la primera búsqueda del ejercicio anterior para sacar el nombre.

- Otra para extraer el nombre únicamente.

3. El día en el que se mandó el email (os deberían salir 2).

Pista De la misma forma que antes buscamos por `From:` primero, ahora lo tendremos que hacer con `Date: .*`

4. El asunto del email (os deberían salir 2).

Pista Busca primero `"Subject:.*"` y después busca el patrón para extraer el asunto.

5. Guarda todos los resultados en un diccionario

BONUS

Utiliza funciones para cada una de los objetivos planteados.

Happy coding!

Recordad que ha modo de apoyo tenemos la página [regex101](#) para probar nuestros patrones de búsqueda.



email.txt 8KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Python + SQL 1

Aprender a conectarse a una base de datos usando Python y a realizar consultas SQL básicas.

Python + SQL 1

En esta lección vamos a aprender a conectarnos a bases de datos y realizar consultas SQL básicas mediante código Python. De esta manera podremos integrar las consultas en nuestros programas para extraer datos almacenados y analizarlos. También podremos guardar los resultados de nuestros programas en tablas de una base de datos.

MySQLConnector/Python

Para todo lo anterior vamos a utilizar la librería de Python llamada MySQL Connector/Python la cual proporciona una serie de clases y métodos que nos permiten acceder a bases de datos MySQL.

Dependiendo de la versión de Python que estemos usando, necesitaremos usar una u otra versión de MySQL Server y su conector Python. Usad <https://dev.mysql.com/doc/connector-python/en/connector-python-versions.html> como referencia para saber que versión debéis utilizar.

Instalación de MySQL Connector/Python

Existen múltiples opciones a la hora de instalar los conectores de MySQL para Python, ya sea descargando y compilando el código fuente, descargando e instalando los binarios ya compilados para el S.O. de nuestra elección, o instalándolo usando un gestor de paquetes como pip. Nosotras usaremos esta última opción al ser la más sencilla. Si queréis más información sobre otras maneras de instalación, podéis consultar <https://dev.mysql.com/doc/connector-python/en/connector-python-installation.html>

Durante el curso hemos estado usando Visual Studio Code, por lo que usaremos la terminal integrada en el mismo para la instalación. Para ello, abre VS Code → Terminal → New Terminal. Esto nos dará acceso a una terminal en la que podemos ejecutar pip para la instalación de paquetes Python:

```
$ pip install mysql-connector-python
```

Es posible que no tengamos instalado pip en VS Code, por lo que de ser necesario, tendremos que ejecutar primero el siguiente comando:

```
$ sudo apt install python-pip
```

Una vez instalado MySQL Connector/Python, si queréis conocer la ruta (path) donde se ha instalado, podéis ejecutar lo siguiente en la terminal:

```
$ python
>>> from distutils.sysconfig import get_python_lib
>>> print(get_python_lib())
```

Por último, para comprobar si MySQL Connector se ha instalado correctamente, podemos ejecutar desde la terminal:

```
$ python
>>> import mysql.connector
```

Si no aparece ningún mensaje de error, significa que la librería se ha instalado correctamente y que ya podemos comenzar a usarla en nuestro código.

Jupyter Notebook

Una vez tengas Python, MySQL Server y MySQL Connector/Python instalados en tu ordenador, pasa a seguir los ejercicios y teoría del siguiente notebook:



modulo-1-leccion-13-python_sql.ipynb 91KB
Binary

Repaso de la lección Python + SQL 1

- Para conectarnos a una BBDD desde Python utilizaremos el método `connect()`
 - Los argumentos a pasarle en el connect son:
 - user: La usuaria con la que nos conectar a la base de datos
 - password: La contraseña de acceso
 - host: La dirección IP (web) de la base de datos
 - database: El nombre de la base de datos a utilizar
 - raise_on_warnings: Si la consulta genera algún aviso, lo que hace es parar la ejecución como si fuera un error.

Definimos el cursor: Siendo el cursor lo que nos permite hacer consultas a la base de datos. Para ello debemos utilizar método.

- `.`cursor()``: Nos permite realizar consultas y nos almacena el resultado de la consulta.

- Para realizar consultas:
 - `execute()` : Nos ejecuta la consulta que le hayamos definido como argumento
 - `fetchone()` : Nos devuelve el primer resultado de la consulta, en forma de tupla.
 - `fetchall()` : Nos devuelve todos los resultados de la consulta, en forma de lista de tuplas.
- Cerrar la conexión a la base de datos despues de la consulta.
 - `close()`
- Si queremos hacer consultas dinamicas, es decir, consultas en la que cambiamos datos a lo largo de ejecución o queremos ejecutar algo dentro del un bucle for podemos realizar de la siguiente forma:

```
query = "SELECT * FROM tabla WHERE col1 = %s"

lista_ciudades = ['Madrid', 'Valladolid', 'Zaragoza']

for values in lista_ciudades:
    cursor.execute(query, values)
```

- Existe un método para guardar los resultados en un dataframe, utilizando la librería de pandas:
 - `pd.read_sql(sql, connector)`

Pair programming

Es el momento de dejar de usar la *interface* de Workbench y empezar a usar SQL desde Python!

Objetivos

1. Antes de empezar a trabajar con la BBDD de Northwind, conozcamos un poco las BBDD que tenemos en nuestro servidor. ¿Qué BBDD tenemos en nuestro servidor?
2. Empezemos a explorar la BBDD de Northwind. ¿Qué tablas componen la BBDD? Recuerda que primero nos tendremos que conectar con la BBDD sobre la que queremos trabajar. Guarda el resultado de los nombres de la tablas en una lista.
3. ¿Qué columnas tiene cada una de las tablas? Para esto tendrás que hacer un for loop para que nos saque el resultado de todas las columnas.
4. Hagamos unas *queries* facítas:
 - Extraed el primer pedido donde se ha gastado más. Devolved el id del pedido y la cantidad gastada.
 - Devuelve la misma *query* que en el anterior ejercicio, pero en este caso devuelve todos los resultados.
 - Extraed el números de pedidos que se hayan hecho por dia. Devuelve los resultados en una *dataframe*.
 - Extraer los pedidos hechos por "Nancy Davolio". Tendréis que hacer una subquery.
5. Para finalizar el ejercicio, desconectad la conexión con el servidor.

Happy coding

Python + SQL 2

Aprender a crear tablas, y modificarlas: INSERT, UPDATE, DELETE, usando Python.

Python + SQL 2

En esta lección vamos a continuar ampliando el conocimiento acerca de MySQL Connector/Python para poder crear tablas y modificarlas dentro de nuestros scripts de Python. Concretamente veremos como crear una tabla usando CREATE, como introducir registros en la misma con INSERT, actualizar registros ya existentes con UPDATE y borrarlos con DELETE.

Jupyter Notebook

Para seguir los ejercicios y la teoría de la lección, descarga y ejecuta el siguiente notebook:



modulo-1-leccion-14-python_sql.ipynb 25KB
Binary

Repaso Python-SQL 2.

- Para hacerlo desde Python:

- Para hacerlo desde Python:

- Importamos la librería

```
import mysql.connector
```

- Conectarnos con la BBDD

```
cnx = mysql.connector.connect(user='root', password='AlumnaAdalab',
                               host='127.0.0.1')
```

- Iniciamos el cursor

```
mycursor = cnx.cursor()
```

- Hacer la query con `execute("IRA LA QUERY")`
- Usaremos el `commit()` para que los cambios se reflejen en la base de datos.
- Para crear BBDD usaremos `CREATE DATABASE`.
- Método `rollback()`: para volver atrás en una query a la que **todavía no la hemos hecho commit**
- `executemany()`: nos permite añadir varios registros a la vez.
- Cuando queremos insertar muchos datos diferentes a nuestra tabla tendremos que usar **una lista de tuplas**
- `%s`: nos permite hacer *queries* dinámicas.
- `rowcount`: me va a contar el número "filas" que se han insertado, o se han modificado, o seleccionado.

- Gestión de errores trabajando con SQL y Python:

- Usamos un `try ... except`
- En `try` especificaremos la query que queremos ejecutar
- En el `except`:

```
except mysql.connector.Error as err:  
    print(err) # printea el error que estamos comentiendo. Es lo mismo que nos sale en  
    print("Error Code:", err.errno) # el número del error.  
    print("SQLSTATE", err.sqlstate) # el estado del error  
    print("Message", err.msg) # el mensaje(texto explicativo) del error.
```

Pair programming

Sigamos trabajando con SQL desde nuestro jupyter! En este caso crearemos una BBDD

Objetivos

Durante la clase de *pair programming* de creación de tablas de SQL nos creamos nuestra primerita BBDD desde cero en Workbench. En el ejercicio de hoy volveremos a crear esa misma BBDD, pero esta vez desde Python.

NOTA Utilizad el nombre de la BBDD de *mi_primerita_BBDD_Python*

NOTA Cuando creeis las distintas tablas de vuestra BBDD repetereis la mismas líneas de código una y otra vez. Para evitar esto, creareis una función que nos sirva para crear tablas independientemente del contenido de la tabla. La función recibirá como parámetros:

- Contraseña de la conexión al servidor.
- *Query* para crear la tabla.

A modo de recap de lo que contenía la BBDD:

- Tabla Zapatillas: tiene 4 columnas: id, modelo, color, talla con las siguientes características:
 - `idZapatillas` : es una clave primaria de tipo int, autoincremental y no nula.
 - `Modelo` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Color` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Talla` : es entero, no nula.
- Tabla Clientes: tiene 9 columnas idcliente, nombre, numero_telefono, email, direccion, ciudad, provincia, pais, codigo_postal con las siguientes características:
 - `idCliente` : es una clave primaria de tipo int, autoincremental y no nula.
 - `Nombre` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Numero_telefono` : es un entero de longitud máxima de 9 caracteres, no nula.
 - `Email` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Direccion` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Ciudad` : es una cadena de caracteres de longitud máxima de 45 caracteres, puede ser nula.
 - `Provincia` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Pais` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Codigo_postal` : entero de máxima longitud 9, no nula.
- Tabla Empleados: tiene 5 columnas idEmpleado, nombre, tienda, salario, fecha_incorporacion con las siguientes características:
 - `idEmpleado` : es una clave primaria de tipo int, autoincremental y no nula.
 - `Nombre` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `Tienda` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `salario` : es decimal, puede ser nula.
 - `fecha_incorporacion` : es una columna de tipo date, no nula.
- Tabla Facturas: tiene 5 columnas idFactura, idEmpleado, idCliente, fecha, total con las siguientes características:
 - `idFactura` : es una clave primaria de tipo int, autoincremental y no nula.
 - `idEmpleado` : es una clave foránea de tipo int, no nula.
 - `idCliente` : es una clave foránea de tipo int, no nula.
 - `idZapatilla` : es una clave foránea de tipo int, no nula
 - `Fecha` : es una columna de tipo date, no nula.
 - `Total` : es decimal, no nula.

NOTA En esta última tabla tendremos que incluir todos los `CONSTRAINT` necesarios para establecer los relaciones entre las tablas, donde tendremos que especificar:

- Foreign Key
- References

Happy coding

Repaso Python I

Repaso I Python Básico

En este apartado tendréis el jupyter con una serie de preguntas que usaremos para la primera sesión de repaso de Python.



repaso_python_1.ipynb 7KB

Binary



repaso_python_1_soluciones.ipynb 31KB

Binary

Repaso Python II

Repaso II Python Básico

En este apartado tendréis el jupyter con una serie de preguntas que usaremos para la primera sesión de repaso de Python.



repaso_python_2.ipynb 6KB

Binary



repaso_python_2_soluciones.ipynb 17KB

Binary

Repaso Python III

Repaso III Python Ficheros y Clases

Aquí encontraréis los ficheros de jupyter con las soluciones de los ejercicios de repaso:

- Soluciones planteadas por los profes:



repaso_python_3.ipynb 15KB

Binary

- Las soluciones planteadas durante la clase de repaso:



repaso_python_3_promo_c.ipynb 12KB

Binary

- Fichero para desarrollar el ejercicio de `xml`:



data.xml 910B

Binary

Repaso Python IV

Repaso IV Python Regex

En el repaso de hoy recordaremos los conceptos básicos de Regex. En este apartado encontraréis dos ficheros:

- Jupyter Notebook donde encontraréis las preguntas que intentaremos contestar durante la clase de repaso
- Un txt con un fragmento del Quijote sobre el que trabajaremos.



repaso_python_4_Regex.ipynb 4KB

Binary



2donq10.txt 2MB

Binary

SQL

Importación de las bases de datos de las clases

En este apartado vamos a explicar como importar las bases de datos que necesitaremos para cada una de las lecciones correspondientes a SQL.

De esta forma se ha creado una base de datos diferente para cada una de las lecciones, ya que de esta forma se podrá probar sobre las tablas ejemplo los ejemplos explicados a lo largo de la teoría.

Así mismo incluye una serie de bases de datos de ejemplo que vienen por defecto con la instalación de MySQL Workbench, en caso de que no se tenga acceso a ellas.

Para ello descarga los siguientes archivos:



create_schemas_clases.sql 528B
Binary



clases_db.zip 846KB
Binary

Ahora para cargar los datos sigue los siguientes pasos:

1. Abre MySQL Workbench y accede al servidor de Adalab Server que creamos durante la instalación de MySQL - MySQL Workbench.
2. Vas a la pestaña de archivo, seleccionas abrir script de SQL y abres **create_schemas_clases**. Tras esto dale al icono del rayo. Si te surge un error que no se puede crear 'sakila' porque la base de datos ya existe por ejemplo, salta esa línea, selecciona las demás y dale al rayo otra vez, hasta el final del archivo (línea 17). Tras esto se habrán creado las bases de datos necesarias para las clases.
3. Ahora vamos a cargar los datos a cada una de las bases de datos, para ello ve a la pestaña de la parte superior llamada **Servidor** y selecciona la opción **Data Import**.
4. Selecciona la opción **Importar desde la carpeta del proyecto de volcado** y selecciona la carpeta donde hayas extraído los archivos de las tablas de las lecciones.
5. Asegúrate que todas las tablas de las bases de datos están marcadas y dale a **Empezar importación**.
6. Al finalizar ya tendrás en tu servidor de MySQL todas las tablas necesarias para las clases!

Introducción a las bases de datos

1.1 Introducción a las bases de datos

Antes de comenzar

A lo largo de estas lecciones se mostrarán ejemplos de código. Te recomendamos sin lugar a dudas, que pruebes dichos ejemplos (más adelante te enseñaremos cómo hacerlo). La idea de estas pequeñas partes de código es que juegues con ellas en tu ordenador y así puedas ver cómo funcionan y probar qué pasaría si las modificas, añadiendo o quitando partes a las mismas, ayudando a que interiorices mejor el aprendizaje. Se trata de descubrir cómo funcionan las bases de datos y el lenguaje SQL en concreto, saber qué se puede hacer con él y qué no.

Introducción a las bases de datos

El objetivo de este curso es convertiros en unas grandes conocedoras de las bases de datos informáticas. Al finalizarlo con éxito vais a ser capaces de diseñar vuestras propias bases de datos, crearlas, modificarlas, consultarlas y eliminarlas. Como todo camino debe comenzar con un primer paso, empezaremos por lo básico. En este capítulo vamos a realizar una primera aproximación teórica a los datos en informática y sus tipos, y a qué son las bases de datos y qué categorías existen. Por último, os presentaremos el modelo relacional, un modelo teórico que se usa comúnmente para diseñar bases de datos: cuáles van a ser sus componentes, su estructura, etc.

¿Qué son los datos?

Los datos pueden ser hechos o valores relacionados con cualquier objeto (real o no). Por ejemplo, el nombre o la edad constituyen un dato acerca de una persona. Otros ejemplos de datos que podemos encontrarnos son una imagen, un video, un archivo de texto, etc.

Tipos de datos

Los datos pueden ser de diversos tipos y naturalezas, por ejemplo, datos numéricos (la edad de una persona o la nota de un examen), letras (un nombre) o una combinación de ambos (un DNI).

Normalmente los datos suelen ser combinaciones de caracteres alfanuméricos (números y letras), pero no siempre es así. Ejemplos de datos más complejos debido a su estructura serían imágenes, videos, páginas web, etc.

Una división comúnmente aceptada para los datos consiste en diferenciar entre datos estructurados y no estructurados:

- **Estructurados:** tienen un formato bien definido. Ejemplos: fecha de nacimiento, DNI, número de cuenta corriente, etc. Son los típicos datos que tienen las empresas almacenadas en tablas o documentos Excel con filas y columnas bien etiquetadas. Se sabe cuantos campos van a tener y que valores pueden tomar.
- **No estructurados:** son la mayoría de los datos que encontramos en el mundo real. No tienen estructura interna bien definida. Normalmente, cuando los datos se pueden almacenar sin tener que especificar su formato significa que estamos tratando con datos no estructurados. Por ejemplo, las imágenes, archivos de video o audio, archivos de texto, etc. contienen datos no estructurados.
- **Semiestructurados:** son los datos que no tienen un formato fijo pero que sí contienen dentro de ellos algo de información sobre su estructura. Los archivos HTML o XML, que son ficheros que contienen texto con etiquetas para definir sus distintas partes serían buenos ejemplos de datos semiestructurados.

Traditionally, the use of face for recognition has achieved lower accuracy compared to fingerprint and iris due to the higher variability in the capture conditions. Nowadays, high accuracy levels can be achieved for face recognition in constrained scenarios, in which the users are collaborative, and the acquisition conditions are favorable. However, some of the most relevant applications of face recognition happen under unconstrained conditions [14], therefore the ability to deal with variability factors needs to be addressed.

The performance of face recognition systems is influenced by the variability of samples [2]. This variability is associated to the image acquisition conditions: illumination, location, background homogeneity, focus, sharpness, etc. Other factors are more related to the properties of the face itself like pose, presence of occlusions, and different expressions. All these factors influence the *quality* of the face samples, which is generally understood as a predictor of the goodness of a given face image to be used for recognition purposes, that is, quality is an estimator of biometric performance.

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>...</head>
<body>
  <div class="header">...</div>
  <script type="text/javascript">...</script>
  <!--<div id="languageBar"><a href="#">en</a> | <a href="#">es</a></div>-->
  <table width="100%">
    <tbody>
      <tr>
        <td class="menu" width="110px">...</td>
        <td class="section">
          <style>...</style>
          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
          <div class="slideshow-container">
            <div class="mySlides fade" style="display: none;">...</div>
            <div class="mySlides fade" style="display: block;">...</div>
            <a class="prev" onclick="plusSlides(-1)"></a>
            <a class="next" onclick="plusSlides(1)"></a>
          </div>
          <br>
          <div style="text-align:center">...</div>
          <script>...</script>
          <!-- NEWS -->
          <div class="news">...</div>
          <!-- -->
        </td>
      </tr>
    </tbody>
  </table>
  <div class="footer">...</div>
</body>
</html>
```

Izquierda: Ejemplo de datos no estructurados (texto libre). Derecha: ejemplo de datos semiestructurados (código HTML)

Como Data Analysts, nuestra función normalmente consistirá en realizar un estudio de los datos de cara a evaluar la información contenida en los mismos y poder tomar decisiones estratégicas con los mismos. Las empresas habitualmente analizan datos estructurados en su día a día ya que es más fácil extraer información de estos. Es por ello por lo que en Adalab nos centraremos principalmente en el trabajo con datos estructurados.

Ejercicio: Piensa 3 ejemplos de datos estructurados, semiestructurados y no estructurados. Indica por qué pertenecen a cada una de las categorías.

¿Qué es una base de datos?

Una base de datos es una colección bien estructurada de datos que presentan ciertas relaciones entre ellos. Las bases de datos buscan tener cierta estructura de cara a facilitar la organización, modificación y recuperación posterior de los datos contenidos en ellas. En el pasado las bases de datos consistían en un montón de archivos en papel guardados en grandes archivadores, pero actualmente las bases de datos informáticas son el estándar en todas las empresas, administraciones, etc., debido a su mayor conveniencia ya que no ocupan tanto espacio y es más fácil recuperar datos e información.

Ejercicio: ¿Se te ocurren ejemplos de bases de datos que utilices en tu día a día? ¿En qué sentido crees que facilitan alguna tarea?

¿Qué tipos de bases de datos hay?

En la vida real existen muchos tipos diferentes de bases de datos. Cada tipo está orientado a una cosa distinta y por lo tanto su diseño y características no serán las mismas. Algunos ejemplos serían:

- Bases de datos que guardan datos importantes, como las de los bancos, que almacenan todos nuestros movimientos bancarios. Están pensadas para ser muy seguras y no perder información bajo ningún concepto.
- Bases de datos pensadas para guardar información efímera. Son más pequeñas y buscar datos en ellas puede hacerse de manera muy rápida. Un ejemplo de este tipo de bases de datos son las que guardan las notificaciones emergentes de WhatsApp de tu móvil. Cuando una amiga te envía un WhatsApp, se añade la notificación a una base de datos. Horas después, tú enciendes el móvil, WhatsApp comprueba que en su base de datos tienes notificaciones pendientes y te las envía. Por último, el servidor de WhatsApp borra la notificación de su base de datos para siempre jamás. Un ejemplo práctico que utilizaremos de forma recurrente durante el curso consiste en una base de datos que contiene información acerca de las alumnas de Adalab. Esta base de datos podría contener los nombres y apellidos de las alumnas, sus emails, contraseñas, calificaciones, etc. También podría contener información acerca de las relaciones entre alumnas (por ejemplo, si son o han sido compañeras en algún proyecto durante el curso).

Ejercicio: Pon un ejemplo de base de datos e indica qué datos podría contener y qué características serían deseables respecto a tamaño, rapidez, seguridad, etc. Teniendo eso en cuenta, ¿a qué categoría de las dos anteriores pertenecería?

Llegadas a este punto ya deberíamos ser capaces de hacernos una primera idea mental sobre cómo son las bases de datos. Podemos pensar en ellas como un conjunto de tablas con filas y columnas, en las que las columnas se corresponderían con los distintos campos de datos, mientras que las filas de la tabla serían el equivalente a las diferentes entradas en las bases de datos.

id	nombre	apellido	email	telefono	direccion	ciudad	pais
1	Ana	González	ana@adalab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@adalab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@adalab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@adalab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@adalab.es	616365624	Calle Alumna 5	Paris	Francia

La tabla ejemplo superior se corresponde con una de las tablas que compondrían la base de datos sobre las alumnas de Adalab. Esta tabla contiene información personal de las alumnas, cada fila correspondiendo a una alumna y cada columna a distintos datos sobre estas (su nombre, DNI, email, etc.). Otra tabla perteneciente a esta misma base de datos podría contener datos sobre los proyectos a realizar/realizados durante el curso, en otra más se podrían guardar los datos de los profesores, etc. Es importante remarcar que una base de datos puede estar compuesta por una o más tablas y que también se pueden definir relaciones entre las distintas tablas, ya que estas a veces comparten datos (pero eso lo veremos más adelante en el curso).

¿Por qué usar bases de datos?

Hoy en día, de cara a tener una utilidad real, casi todo programa informático necesita cargar y guardar datos, ya sean propios de la aplicación o datos externos. Para conseguir esto, se almacenan los datos en bases de datos donde estarán disponibles para su posterior consulta.

Cuando trabajamos con una base de datos, los cambios que hacemos sobre ella se guardan en ficheros. Gracias a ello, si se reinicia el servidor y se vuelve a abrir la base de datos, esta mantendrá los datos que tenía la última vez que se guardó. De esta manera las bases de datos nos permiten guardar datos de forma permanente. Si los datos que se quieren guardar son pocos y sin estructura, se pueden utilizar archivos "normales" (texto plano) para su almacenamiento. Sin embargo, cuando se necesita manejar grandes volúmenes de datos y además estos datos están relacionados unos con otros, utilizar archivos de texto plano resulta inviable siendo poco eficiente y mucho más complicado recuperar los datos a posteriori.

Las bases de datos surgen como solución a esta necesidad de una manera más eficiente de guardar y acceder a los datos. Los principales beneficios de su uso serían:

- **Evitar inconsistencias entre los datos** impidiendo que exista información discrepante sobre un único hecho, persona o entidad.
- **Reducción de la redundancia en los datos.** De esta manera se evita almacenar múltiples veces un mismo dato, reduciendo así el almacenamiento necesario para guardarlos.
- **Compartir los datos** entre distintas usuarias y aplicaciones, gestionando el acceso simultáneo a la información sin que se produzcan conflictos.
- **Garantizar la seguridad** de la información, controlando el acceso y la manipulación de estos por parte de las distintas aplicaciones y usuarias.

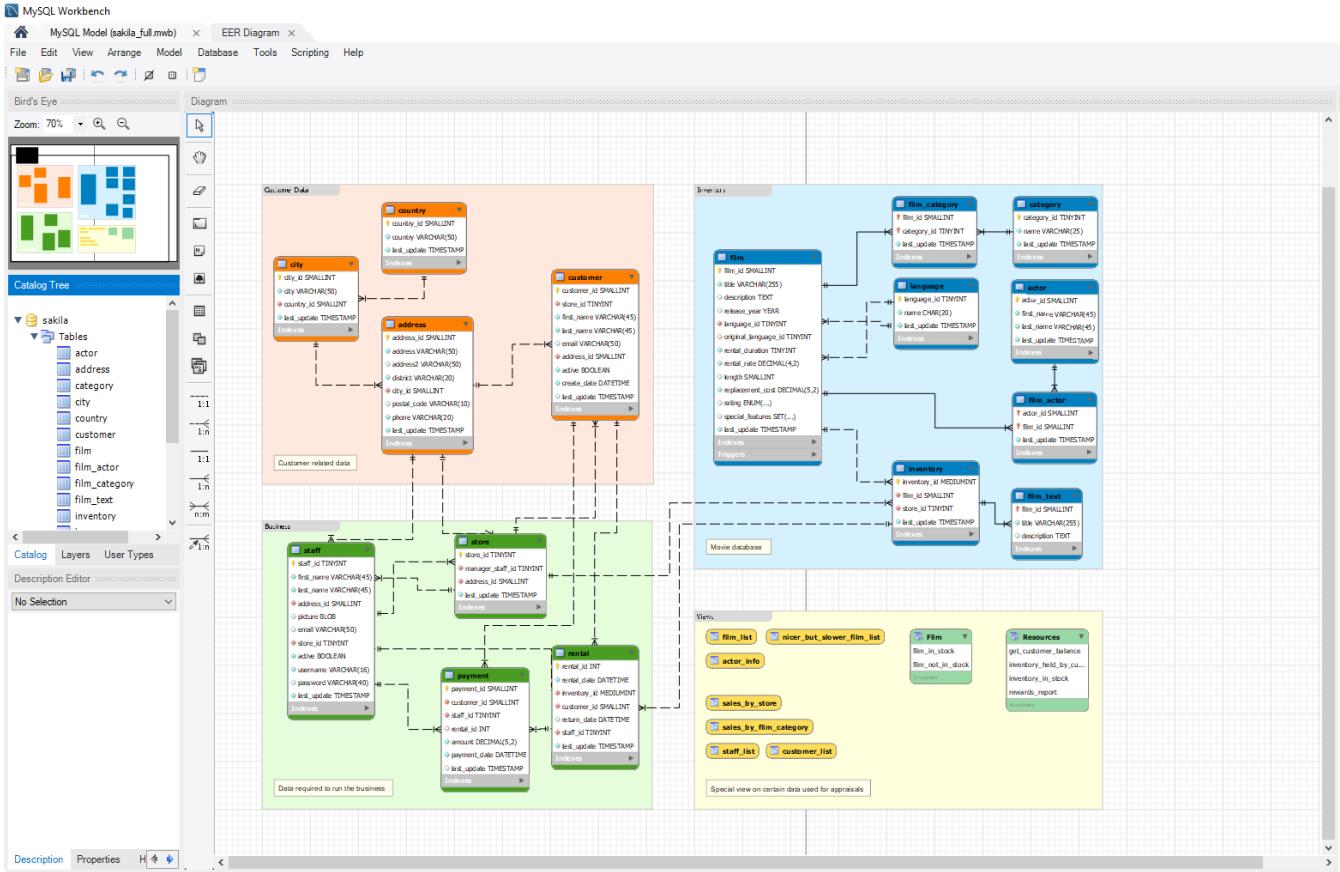
Ejercicio: Teniendo en cuenta los beneficios de las bases de datos, ¿qué aplicaciones se te ocurrirían para las mismas? Piensa en dos o tres casos concretos en los que usar bases de datos pueda ser interesante.

¿Qué es un Sistema de Gestión de Bases de Datos (SGBD)?

Las administradoras y usuarias de las bases de datos (vosotras en un futuro cercano) deben ser capaces de acceder fácilmente a estas de cara a poderlas crear, manipular, mantener y actualizar. Esto se consigue mediante programas informáticos denominados Software de Gestión de Bases de Datos (SGBD).

Un SGBD (o DataBase Management System - DBMS por sus siglas en inglés) puede gestionar una o más bases de datos permitiendo explorar los datos almacenados en ellas de forma más o menos simple usando consultas en Structured Query Language (SQL), un lenguaje informático diseñado específicamente para crear, consultar, modificar, eliminar y actualizar datos en una base de datos.

Durante el presente curso estudiaremos en profundidad el lenguaje SQL para crear y trabajar con bases de datos. Como SGBD usaremos MySQL, que es multiplataforma y de código abierto, lo que significa que se puede ejecutar en varios tipos de sistemas operativos diferentes, como Windows, Linux y Mac OS, y que puede consultarse y modificarse su código interno para añadir nuevas funcionalidades o modificar las existentes. Esto los hace diferentes a los SGBD propietarios (y a menudo preferibles), que son gestionados y licenciados por compañías, las cuales ofrecen un paquete cerrado de características con posibilidades limitadas de modificación.



Ejemplo de SGBD: MySQL Workbench. Se muestra el esquema de una base de datos (tablas, relaciones, etc.)

¿Dónde se ejecutan las bases de datos?

Las bases de datos se ejecutarán en un ordenador que denominaremos servidor. Dicho servidor puede ser dedicado, como un servidor remoto creado a tal efecto al que accederemos mediante Internet o algún otro protocolo, o puede ser nuestro propio equipo.

Repasso base de datos

- Qué es una base de datos y como se estructuran.
- Tipo de base de datos: SQL y noSQL
- Tipo de datos: Estructurados, Semiestructurados y no estructurados.
- Qué es un Sistema de gestión de bases de datos (SGBD)

Modelo Relacional

1.2 Modelo Relacional

Introducción a las bases de datos

Modelo Relacional:

A nosotras nos interesa saber los tipos de bases de datos que existen en función de cómo estructuran los datos. Las bases de datos se pueden organizar siguiendo diferentes modelos. El más comúnmente usado es el modelo relacional, por lo tanto, será en el que usaremos en Adalab.

El modelo relacional es un modelo teórico que se utiliza para diseñar bases de datos. En este modelo, las bases de datos son un conjunto de tablas, cada una con una serie de columnas y filas. Volviendo al ejemplo de la sección anterior, en nuestra base de datos podemos tener una tabla llamada `alumnas` con todos los datos de cada una de las alumnas de Adalab.

id_alumna	nombre	apellido	email	telefono	dirección	ciudad	país
1	Ana	González	ana@adalab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@adalab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@adalab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@adalab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@adalab.es	616365624	Calle Alumna 5	Paris	Francia

- Las columnas corresponderían a los atributos de cada alumna, por ejemplo, su nombre, email o dirección.
- Las filas corresponderían a cada una de las alumnas. Cada fila se denomina registro y corresponderá a una alumna diferente.

Las bases de datos se dividen en dos elementos principales: el **esquema** y los **datos**. El **esquema** es la definición de la estructura de la base de datos. Como hemos introducido anteriormente, en una base de datos relacional los datos están representados en tablas, por lo que el esquema contiene normalmente el nombre de cada tabla, el de cada columna y el tipo de dato de cada columna. Por otro lado, los **datos** se corresponden con la información contenida dentro de la base de datos en un momento dado.

customer	
!	customer_id SMALLINT
◆	store_id TINYINT
◆	first_name VARCHAR(45)
◆	last_name VARCHAR(45)
◆	email VARCHAR(50)
◆	address_id SMALLINT
◆	active BOOLEAN
◆	create_date DATETIME
◆	last_update TIMESTAMP

Ejemplo del esquema de una tabla en MySQL. Se indican los atributos y el tipo de dato que será cada uno de ellos.

Ahora que ya tenemos una idea mental aproximada sobre qué es una base de datos, para qué puede servirnos y como sería su estructura, podemos entrar a describir más en profundidad los diferentes elementos que las componen, siguiendo una terminología más precisa:

- **Tablas:** representaciones bidimensionales de los datos compuestas por filas/registros y columnas/atributos. Ejemplo: una tabla sobre las alumnas de Adalab.
- **Atributos:** son las columnas de la tabla. Contienen algún dato perteneciente a cada registro. Por ejemplo, el DNI o el nombre de las alumnas serían algunos de los atributos.
- **Registros:** son las filas de las tablas. Representan un elemento único del tipo de datos considerado en la tabla. Contienen valores para cada uno de los atributos definidos. En nuestro ejemplo serían las entradas para cada una de las diferentes alumnas.
- **Claves:** atributos cuyo valor es único para cada registro. No se pueden repetir. Sería la columna ID de la tabla que hemos puesto como ejemplo donde cada alumna tiene el suyo propio.
- **Grado:** el grado de una tabla es el número de atributos que contiene.

Ejercicio: Piensa en un ejemplo de tabla para una base de datos que se pudiese usar en sitios web como Amazon o similares. Puede ser información sobre clientes, productos, envíos, etc. Indica qué atributos tendría dicha tabla, cuál sería su grado y qué atributos podrían utilizarse como clave.

Según los valores que pueda tomar cada atributo, se puede distinguir entre diferentes tipos:

- **Monovaluados o multivaluados:** los atributos monovaluados solo pueden tomar un valor, por ejemplo, cada alumna solo tiene un DNI, por lo que este es un atributo monovaluado. En cambio, los atributos multivaluados pueden tomar varios valores a la vez. Este sería el caso del atributo “Titulación” de una alumna, ya que cada alumna podría tener más de una titulación.
- **Simples o compuestos:** si el valor del atributo no se puede dividir en campos más sencillos, se considera simple. Un ejemplo de esto sería la edad de las alumnas. Por el contrario, la dirección de las alumnas sería un atributo compuesto ya que puede dividirse en calle, número, piso, ciudad, país, código postal, etc.
- **Almacenados o derivados:** los atributos almacenados son aquellos que están realmente guardados en la base de datos. Sin embargo, un atributo derivado sería aquel que no se guarda ya que se puede deducir a partir otros atributos almacenados. Ejemplos de estos tipos de atributo serían la fecha de nacimiento (almacenado), y la edad (derivado). Es importante que se decida sabiamente qué datos van a ser almacenados y cuáles serán derivados para evitar problemas futuros. Por ejemplo, la fecha de nacimiento no cambia nunca por lo que es un buen candidato a atributo almacenado. Sin embargo, la edad no interesa guardarla en la base de datos ya que se puede deducir de otros atributos y debería actualizar cada año.

Ejercicio: En la tabla que pensaste en el ejercicio anterior, clasifica sus atributos para cada una de las categorías que acabamos de ver: mono o multivaluados, simples o compuestos, y si deberían ser almacenados o derivados. Razona la respuesta.

Dominio de un atributo:

Es el conjunto de valores que puede tomar un atributo concreto. En el caso del DNI de las alumnas, el dominio sería: 8 dígitos + 1 letra.

El dominio de cada atributo tiene que ser definido en el momento de diseño de la base de datos (como veremos más adelante).

Clave primaria (Primary Key):

El modelo relacional asegura que cada registro de una tabla sea único, es decir, que no se repitan los datos. Por lo tanto, dentro de cada tabla se debe elegir un atributo para identificar únicamente cada registro o entrada que introduzcamos en la misma. Es decir, no puede haber en la tabla dos registros que comparten la misma clave primaria. El DNI de las alumnas sería (una vez más) una buena elección de clave primaria, ya que no habrá dos alumnas con el mismo DNI. El ISBN de un libro sería también un buen ejemplo de candidato a clave primaria.

Ejercicio: Siguiendo con la tabla que has diseñado en los ejercicios anteriores, indica el dominio de cada uno de los atributos y cuál de ellos usarías como clave primaria de la tabla.

Clave Foránea (Foreign Key):

Dentro de la base de datos, las entradas de cada tabla pueden estar relacionada con entradas en otras tablas. Esta relación se indica mediante el uso de atributos definidos como claves foráneas, o foreign keys.

La clave foránea identifica una columna en una tabla (la tabla referendo) que se corresponde con una columna de otra tabla (la tabla referenciada). Las columnas de la tabla referendo deben ser la clave primaria u otra clave de la tabla referenciada, es decir, los valores de la clave foránea para cada fila de la tabla referendo deberán existir en una única fila de la tabla referenciada.

id_alumna	nombre	apellido	email	telefono	dirección	ciudad	pais
1	Ana	González	ana@ad.alab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@ad.alab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@ad.alab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@ad.alab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@ad.alab.es	616365624	Calle Alumna 5	Paris	Francia

En la tabla `alumnas` (mostrada sobre estas líneas) la clave primaria será el atributo `id_alumna`, ya que es un número asignado a cada alumna de manera individual y que no se repite. Email y telefono podrían ser otros candidatos a clave primaria.

<code>id_proyecto</code>	<code>id_alumna</code>
1	1
2	5
3	4
4	2
5	3

En la tabla superior llamada `AlumnasYProyectos`, perteneciente a la misma base de datos que `alumnas`, se está indicando a qué proyecto está asignada cada alumna de la base de datos. Aquí la columna `IDAlumna` podría ser una clave foránea válida para esta tabla ya que está haciendo referencia a una columna que es clave en otra tabla (columna `ID` en la tabla `alumnas`).

Hay que remarcar que la tabla referendo y la tabla referenciada pueden ser la misma. Esto significa que la clave foránea hace referencia a otras entradas (registros) de la misma tabla. A su vez, una tabla puede tener múltiples claves foráneas, cada una pudiendo hacer referencia a una tabla distinta.

Debido a todas las restricciones impuestas a los valores de las claves en las bases de datos, a la hora de actualizar una tabla (introduciendo o eliminando datos) se garantiza la integridad de los datos, cosa que no sería posible simplemente guardando los datos en ficheros de texto plano.

Cardinalidad de las relaciones:

Cuando se construye una base de datos, la cardinalidad suele referir al número de filas en la tabla A que se relacionan con la tabla B.

Los distintos tipos de cardinalidad son:

- **1:1:** relación uno a uno. Cada entrada en la tabla A se refiere a una sola entrada en la tabla B y viceversa. Por ejemplo, en la base de datos de recursos humanos de una empresa, se puede tener una tabla para las empleadas y otra tabla para los departamentos. En este caso, cuando una empleada dirige un solo departamento en una empresa y cada departamento está dirigido por una sola empleada, se daría una relación 1:1.
- **1:N:** relación uno a muchos. Una entrada en la tabla A se relaciona con varias entradas en la tabla B. En el ejemplo de la empresa que venimos utilizando, este tipo de relación podría darse cuando cada empleada pertenece a un único departamento, pero cada departamento tiene muchas empleadas.
- **M:N:** relación muchos a muchos. Volviendo al mismo ejemplo, sería el caso en el que una empleada trabaja en muchos proyectos, a la vez que cada proyecto tiene muchas empleadas.

Ejercicio: Piensa en una base de datos con dos o tres tablas al menos (puedes volver a pensar en una típica BD que se usaría en sitios como Amazon: clientes, productos, envíos, etc.). Indica las relaciones que deberían existir entre las tablas, su cardinalidad y qué atributos estarían relacionados en cada caso.

Participación en las relaciones:

Dependiendo de si todas las entidades de un mismo tipo tienen que participar en una relación concreta, esta se clasificará como total o parcial. En el ejemplo de la empresa, si cada empleada tiene que pertenecer obligatoriamente a un departamento, esta sería una participación total. Si por el contrario pudiese haber alguna empleada que no perteneciese a un departamento (quizás la directora general), entonces se trataría de una participación parcial.

Ejercicio: De las relaciones del ejercicio anterior, ¿cuáles serían parciales y cuáles totales?.

Repasso Modelo Entidad-Relacion

- Terminología de bases de datos:
 - Tablas
 - Atributos o columnas
 - Registros o filas
 - Grados
- Cardinalidad de las relaciones entre las tablas:
 - 1:1
 - 1: N
 - N: M
- Clave primaria y foránea.
 - La clave foránea siempre será clave primaria de otra tabla.
- Valores de los atributos:
 - Monovaluados y multivaluados
 - Simples y compuestos
 - Almacenados y derivados
- Dominios de atributo

Modelo relacional Pair programming

Pair programming SQL I

Ejercicio para el pair programming

Hemos creado este ejercicio para que lo hagáis durante la hora de pair programming entre tu compañera y tú. Este ejercicio es incremental, es decir, cada día vamos a ir añadiendo las nuevas funcionalidades que hemos aprendido.

Es obligatorio que lo hagáis en la hora de *pair programming* porque:

- Aquí os enseñamos trucos y buenas prácticas.
- Es un ejercicio de tamaño medio, como el que haréis en las entrevistas de trabajo. Aprenderás cosas que no se explican en el resto de materiales.
- El último día del módulo haréis una evaluación sobre este ejercicio con vuestra profesora. Esta evaluación consistirá en una entrevista técnica similar a la de los módulos anteriores, pero en esta ocasión por parejas.

Enunciado

Durante la ejecución de estas sesiones de *pair programming* vamos a trabajar sobre la base de datos Northwind. Esta base de datos se basa en una empresa ficticia: Northwind Traders. Contiene información acerca de las transacciones de ventas entre la empresa y sus clientes así como detalles acerca de compras de la empresa a sus proveedores. Los datos que incluye la base de datos son: tablas de inventario, pedidos, clientes, información de las empleadas, etc. Históricamente ha sido la base de muchos tutoriales y libros sobre el aprendizaje y el uso de SQL.

Base de Datos Northwind



northwindMySQL.sql 225KB
Binary

Fuente: <https://documentation.alphasoftware.com/>

Ejercicios

1. Descarga la base de datos Northwind.
2. Crea el schema con nombre `northwind` en MySQL Workbench para importar la base de datos:
 - Abrid MySQL Workbench y accedid al servidor de Adalab Server que creamos durante la instalación de MySQL - MySQL Workbench.
 - Id a la pestaña de archivo, seleccionad **abrir script de SQL** y abre el archivo NorthwindMySQL.sql que hemos descargado en el paso 2. Tras esto dale al icono del rayo para ejecutarlo. Al finalizar la ejecución se habrán creado las tablas necesarias para los ejercicios de *pair programming*.
3. Explora la estructura de la base de datos:
 - La base de datos Northwind está compuesta por múltiples tablas con relaciones entre ellas. En este momento del curso lo más probable es que no dispongáis de los conocimientos necesarios para inspeccionar el script de creación de la base de datos y entenderlo. Aún así, os invito a que lo leais tranquilamente intentando haceros una idea de qué hace cada una de las instrucciones. Este ejercicio es interesante, porque al finalizar el curso podréis volver a hacerlo y os daréis cuenta de lo mucho que habéis avanzado.
 - Una vez importada la base de datos, podéis utilizar MySQL Workbench para ir viendo qué tablas componen la base de datos y qué columnas/atributos tiene cada una de ellas. Así podréis interiorizar la estructura de Northwind y más adelante tendrás más facilidad a la hora de diseñar las consultas SQL sobre la base de datos.
4. Entiende el objetivo de los ejercicios

En los próximos ejercicios te vamos a guiar paso a paso para que los consigas terminar. Lo importante no es que sigas los pasos de manera automática, sino que entiendas lo que estamos haciendo en cada uno de ellos y por qué. De esta manera será más fácil que generalicéis los conocimientos adquiridos sobre esta base de datos a otras con las que trabajéis en un futuro.

Conclusión

Vuestro objetivo es realizar cada uno de los ejercicios que vayamos presentando durante los próximos días. Para ello tendréis que:

- Analizar lo que os estamos pidiendo en el enunciado general.
- Analizar los datos que se están pidiendo en cada uno de los ejercicios concretos.
- Diseñar la consulta (o consultas) SQL que sea necesaria en cada caso.
- Ejecutar la consulta en MySQL Workbench y analizar los resultados obtenidos.
- Subir el código SQL y los resultados a vuestro repositorio de GitHub.

Indicadnos qué ejercicios habéis hecho

Para llevar un mejor seguimiento de cuál es vuestra evolución durante este módulo os pedimos que en el README de vuestro repositorio en GitHub marquéis con una X los ejercicios hayáis conseguido terminar.

Creación de tablas

Explicaciones SQL y Creación de tablas: CREATE TABLE y tipos de variables

Structured Query Language (SQL)

En este módulo vamos a conocer y a aprender a utilizar Structured Query Language (SQL), un lenguaje informático diseñado para crear, consultar y modificar bases de datos.

¿Qué es SQL?

Como su propio nombre indica SQL (Structured Query Language, en inglés) es un lenguaje estructurado de consulta. Esto significa que fue creado especialmente para administrar la información de las bases de datos relacionales, desde su estructura a los datos contenidos en la misma.

SQL nos permite realizar tareas como:

- Crear nuevas bases de datos.
- Modificar la estructura de la base de datos (creando o eliminando tablas, por ejemplo).
- Insertar nuevos datos en la BD.
- Consultar los datos contenidos en la base de datos.
- Reorganizar y actualizar los datos en la BD.
- Eliminar datos ya existentes.

Hoy en día es el estándar para la gestión de los datos en este tipo de bases de datos. Sin embargo, hay diferentes versiones de SQL cada una con ciertas características especiales, aunque todas esas versiones deben soportar al menos los comandos más importantes, los cuales estudiaremos en las siguientes secciones.

SQL puede utilizarse bajo diferentes Sistemas de Gestión de Bases de Datos (SGBD) como Oracle, SQLite, Posgres o MySQL (que es el que vamos a utilizar en este curso). Las distintas versiones de SQL suelen venir dadas por extensiones que añaden cada uno de estos diferentes SGBD, por lo que algunos comandos usados en SQLite pueden no ser compatibles con MySQL. Esto será importante a la hora de buscar documentación y ayuda en Internet.

¿Dónde se usa SQL? Utilidad del lenguaje

Por ejemplo, si desarrollásemos una página web para realizar compras online (tipo Amazon), los datos de los diferentes productos, sus precios, etc. estarán guardados en una base de datos a la que las usuarias de la web irán realizando consultas. Estas consultas son “invisibles” para la usuaria, ya que son introducidas en el código de la página web por su desarrolladora. De esta manera, cuando la usuaria hace clic en un producto para entrar en su página, el navegador web realiza una consulta a la base de datos de la web, coge los datos y los presenta a la usuaria.

Lenguaje de Manipulación de Datos - DML

En inglés, el Data Manipulation Language (DML), es la parte del lenguaje SQL que permite a la usuaria introducir datos en la base de datos, realizar consultas sobre los mismos y/o modificarlos.

A tal efecto, los comandos SQL más comunes para realizar estas tareas son:

- **SELECT** : esta sentencia SQL se utiliza para realizar consultas sobre los datos.
- **INSERT** : con este comando podemos introducir datos en la base de datos.
- **DELETE** : sentencia usada para eliminar filas en una tabla.
- **UPDATE** : este comando permite modificar los valores de uno o varios registros en una tabla.

En los siguientes puntos de este módulo veremos más en detalle cada una de estas sentencias y realizaremos ejercicios en las que los aplicaremos en nuestra propia base de datos de pruebas.

Data Definition Language (DDL)

El Lenguaje de Definición de Datos es la parte del lenguaje SQL que se utiliza para crear las tablas de las bases de datos, definir los tipos de datos, las relaciones entre las tablas, e insertar y/o modificar datos en las mismas.

Esta parte del lenguaje es la que más varía de un sistema de gestión de bases de datos a otro ya que se está definiendo la estructura y organización interna de los datos en la base de datos, y dicha organización cada sistema lo hace de una manera u otra.

CREATE TABLE

La sentencia `CREATE TABLE` sirve para definir la estructura de una tabla en la base de datos, es decir: qué columnas tiene, sus tipos de dato, relaciones, restricciones, etc.

La sintaxis básica de `CREATE TABLE` es la siguiente:

```
CREATE TABLE nombre_tabla(  
    nombre_columna1 tipo_de_datos1 restriccion1,  
    nombre_columna2 tipo_de_datos2 restriccion2,  
    restriccion3)
```

- `nombre_tabla` : el nombre de la tabla que estamos definiendo con `CREATE TABLE` .
- `nombre_columna1` : el nombre de la primera columna de la tabla.
- `tipo_de_datos1` : el tipo de datos de la primera columna de la tabla.
- `restriccion1` : una restricción para la primera columna de la tabla.
- `restriccion2` : una restricción para la segunda columna de la tabla.
- `restriccion3` : restriccion fuera de columna. Puede afectar a una, varias o todas las columnas de la tabla.

Para escribir una sentencia `CREATE TABLE` se empieza por indicar el nombre de la tabla que queremos crear y a continuación entre paréntesis se deben enumerar las distintas columnas que queremos que compongan la tabla. Las definiciones de las columnas irán separadas con comas.

Para cada columna se deben indicar su nombre, el tipo de dato que contendrá y de manera opcional una serie de restricciones que deberán cumplir los datos almacenados en dicha columna. Adicionalmente, al terminar de definir las columnas que componen la tabla se pueden añadir más restricciones que afectarán a una, varias o todas las columnas. Debido a eso, la sintaxis de este último tipo de restricciones será ligeramente diferente.

Tipos de datos

Cuando se trabaja con bases de datos, los tipos de datos ayudan a conseguir que los datos introducidos en las tablas mantengan el formato esperado. Así se evita que en formularios online por ejemplo, los usuarios introduzcan valores no válidos en los distintos campos. Además, más adelante pueden querer realizarse operaciones con los datos guardados y si no siguen un formato concreto, dichas operaciones pueden ser erróneas.

Algunos de los tipos de datos básicos que resulta necesario conocer si se quieren definir tablas en SQL (o trabajar con ellas) son:

Tipos de datos numéricos:

- `SMALLINT` : un número entero que ocupa 16 bits de almacenamiento (2 Bytes).
- `MEDIUMINT` : un número entero que ocupa 24 bits de almacenamiento (3 Bytes).
- `INT / INTEGER` : un número entero de hasta 10 dígitos. Ocupa 32 bits de almacenamiento (4 Bytes).
- `BIGINT` : Entero de hasta 19 dígitos. Ocupa 64 bits de almacenamiento (8 Bytes).
- `FLOAT` : número decimal con 7 dígitos de precisión decimal. Ocupa 32 bits de almacenamiento (4 Bytes).
- `DOUBLE` : número decimal con 15 dígitos de precision decimal. Ocupa 64 bits de almacenamiento (8 Bytes).
- `BOOL / BOOLEAN` : utilizado para comprobaciones True-False. Un 0 es considerado False y cualquier otro valor es True.

En todos estos tipos de datos se pueden especificar los tamaños de los campos. Por ejemplo: `INT(6)` creará una variable de tipo `INT` con 6 dígitos.

Todos los datos numéricos tienen una opción adicional que puede tomar el valor `UNSIGNED` o `ZEROFILL`. Si añadimos la opción `UNSIGNED` no se permitirán valores negativos. La opción `ZEROFILL` se usa para llenar con ceros a la izquierda todos los espacios disponibles. Además, con `ZEROFILL` se asigna automáticamente el atributo `UNSIGNED` a la columna, por lo que cualquier valor negativo ingresado en una columna con `ZEROFILL` no será válido.

Tipos de datos de texto:

- `CHAR(tamaño)` : Cadena de caracteres de una longitud fija especificada entre paréntesis. Se permiten hasta 255 caracteres.
- `VARCHAR(tamaño)` : Cadena de caracteres de tamaño variable. Se especifica el máximo tamaño que podrá tener. Se permiten hasta 65,535 caracteres.
- `BINARY(tamaño)` : igual que `CHAR` pero almacena cadenas de caracteres binarios.
- `VARBINARY(tamaño)` : igual que `VARCHAR` pero almacena cadenas de caracteres binarios.
- `TINYTEXT` : almacena una cadena de caracteres con una longitud máxima de 255 caracteres.
- `TEXT` : almacena una cadena de caracteres con una longitud máxima de 65,535 caracteres.
- `MEDIUMTEXT` : almacena una cadena de caracteres con una longitud máxima de 16,777,215 caracteres.
- `LONGTEXT` : almacena una cadena de caracteres con una longitud máxima de 4,294,967,295 caracteres.
- `ENUM(val1, val2, val3, ...)` : una cadena de caracteres que puede tomar un solo valor de los indicados en la lista. En dicha lista pueden indicarse hasta 65,535 valores posibles. Si se intenta introducir un valor que no este en la lista, se insertará un valor en blanco.

Tipos de datos de fecha:

- `DATE` : Fecha con formato AAAA-MM-DD. El intervalo permitido va desde el '1000-01-01' al '9999-12-31'.
- `TIME` : Hora con formato HH:MM:SS. El intervalo soportado va de '-838:59:59' a '838:59:59', es decir, cualquier hora posible.
- `DATETIME` : Fecha y hora con formato AAAA-MM-DD HH:MM:SS. El intervalo permitido va desde el '1000-01-01 00:00:00' al '9999-12-31 23:59:59'. Es una combinación de los tipos de datos `DATE` y `TIME`.
- `TIMESTAMP` : Un timestamp es una representación de la fecha y hora actual. El formato es: AAAA-MM-DD hh:mm:ss. El intervalo permitido va desde '1970-01-01 00:00:01' UTC hasta el '2038-01-09 03:14:07' UTC. La diferencia con `DATETIME` es que a la hora de insertar y seleccionar este tipo de datos en una base de datos gestionada por MySQL es que el gestor realiza la conversión de la fecha de tu zona horaria a UTC y viceversa automáticamente.
- `YEAR` : Un año en formato de cuatro dígitos. Los valores permitidos van desde 1901 a 2155 (aunque el 0000 también es un valor admitido).

Existen otros tipos de datos más especiales que pueden encontrarse en [este link](#), aunque estos son los principales.

Ejemplo: Para crear una tabla "productos" podríamos usar la siguiente secuencia:

```
CREATE TABLE productos (
    id_producto INT,
    nombre VARCHAR(100),
    color ENUM('rojo','amarillo','azul'),
    precio INT,
    stock FLOAT
);
```

La tabla tendría las siguientes columnas:

- `id_producto` : El identificador de cada producto, un número entero.
- `nombre` : El nombre del producto. Es una cadena de caracteres de longitud variable de hasta 100 caracteres.
- `color` : El color del producto. Lo hemos definido como una cadena de caracteres que tiene que tomar un valor entre los especificados (rojo, amarillo o azul).
- `precio` : El precio del producto. Será un número entero.
- `stock` : El número de unidades del producto. Puede ser un número decimal (por si hay mitades).

Ejercicio: Crea la tabla Empleadas que hemos estado utilizando en las lecciones anteriores. De momento no te preocupes por definir restricciones a los datos que se pueden introducir, o por definir claves primarias o foráneas. Puedes ver a continuación un ejemplo de la tabla:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Restricciones de columna

Este tipo de restricciones se utilizan para indicar ciertas características o condiciones que se deben cumplir en la columna que se está definiendo en ese momento. Algunas de las cláusulas que se pueden utilizar para este tipo de restricciones son:

- `NOT NULL` : sirve para indicar que la columna en cuestión no puede dejarse vacía (o contener un valor nulo) cuando se inserten entradas en la tabla.
- `PRIMARY KEY` : se usa para indicar que la columna servirá como clave principal de la tabla. Esto conlleva que la tabla no podrá contener valores nulos ni puede haber dos entradas con el mismo valor para este atributo. En la tabla sólo puede haber una clave principal, por lo que no se puede incluir esta cláusula en más de una columna.
- `UNIQUE` : esta cláusula se utiliza para definir una columna como índice único. Esto significa que el atributo en cuestión no permite que haya valores duplicados en la tabla (de la misma manera que la clave principal). Se suele emplear para comprobar que no se están introduciendo entradas que ya existen en la tabla, por ejemplo, definiendo el DNI de una persona como `UNIQUE` .
- `CONSTRAINT` : Esta cláusula es opcional. Sirve para poner nombre a las restricciones, que deben de cumplir los datos. En caso que estemos creando tablas con referencias de claves foráneas, deberemos obligatoriamente añadir la constraint para que se puedan referenciar los datos correctamente. Más adelante en la lección veremos como se realiza esto.
- `REFERENCES tabla [(columna)]` : se usa para definir columnas como claves foráneas de la tabla. Con `REFERENCES` se pueden indicar con qué columnas de qué tablas se corresponde esta clave foránea. Si no se especifica la columna de la tabla externa, se asume que se corresponde con su clave primaria.
- `CHECK (expresion condicional)`: sirve para asegurarse de que los valores en una columna cumplen una determinada condición. Por ejemplo se puede usar para indicar que la edad de una persona debe estar por encima de los 18 años, si queremos que sólo se puedan registrar personas mayores de edad.
- `DEFAULT` : sirve para establecer un valor por defecto para la columna. Dicho valor es el que se le dará al atributo cuando se añada un nuevo registro a la tabla que no contenga valor para esa columna.

Ejemplo: Para crear una tabla "productos" podríamos usar la siguiente secuencia:

```

CREATE TABLE productos2 (
    id_producto INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(100) DEFAULT NULL,
    color ENUM('rojo','amarillo','azul') DEFAULT NULL,
    precio INT DEFAULT NULL,
    stock FLOAT DEFAULT NULL
);

```

La tabla tendría las mismas columnas que en el ejemplo anterior. Se le han añadido las siguientes restricciones de columna:

- `id_producto` : No puede ser `NULL` y se autoincrementa de manera automática si no se especifica valor.
- `nombre` : Si se deja vacío tomará el valor `NULL`.
- `color` : Si se deja vacío tomará el valor `NULL`.
- `precio` : Si se deja vacío tomará el valor `NULL`.
- `stock` : Si se deja vacío tomará el valor `NULL`.

Ejemplo: Creamos la tabla "personas" para contener información de las empleadas de una empresa:

```

CREATE TABLE personas (
    id INT NOT NULL,
    apellido VARCHAR(255) NOT NULL,
    nombre VARCHAR(255),
    edad INT,
    ciudad varchar(255) DEFAULT 'Madrid'
);

```

Restricciones de tabla

- `PRIMARY KEY` (lista de columnas): Es interesante utilizarla como restricción de tabla en vez de restricción de columna cuando se quiere crear una clave primaria que sea la combinación de varias columnas.
- `UNIQUE` (lista de columnas): De la misma manera que con la primary key, se utiliza como restricción de tabla cuando se quieren indicar varias columnas al mismo tiempo para ser `UNIQUE`.
- `FOREIGN KEY` (lista de columnas) `REFERENCES Tabla(columnas)` : Similar a las anteriores, útil para definir varias columnas como claves foráneas al mismo tiempo.
- `CHECK` : En esta ocasión se pueden hacer chequeos de los valores de cualquiera de las columnas de cada registro, así como de varias al mismo tiempo.

Ejemplo: Para crear una tabla "productos" podríamos usar la siguiente secuencia:

```

CREATE TABLE productos3 (
    id_producto INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(100) DEFAULT NULL,
    color ENUM('rojo','amarillo','azul') DEFAULT NULL,
    precio INT DEFAULT NULL,
    stock FLOAT DEFAULT NULL,
    PRIMARY KEY (id_producto),
    CONSTRAINT precio_positivo CHECK ((precio > 0))
);

```

La tabla tendría las mismas columnas que en el ejemplo anterior. Con las mismas restricciones de columna. Adicionalmente se han añadido 2 restricciones de tabla:

- Como `PRIMARY KEY` se ha decidido establecer la columna `id_producto`. Esta columna ya había sido definida con el parámetro `NOT NULL`, pero esto hace que tenga que ser también `UNIQUE`.
- También se ha añadido una restriccion de tipo `CHECK` para comprobar que el precio de un producto tenga que ser mayor que 0.

Ejercicio: Crea la misma tablas Personas que hemos definido anteriormente pero añadiendo una restricción de tabla que comprueba si la edad de la persona es mayor de 16 años. Llámala Personas2.

Ejercicio: Crea la tabla Pedidos, estableciendo restricciones de tabla para la clave primaria y la clave foránea.

Nota :Se ha establecido el ID de pedido como clave primaria y el ID de las personas como clave foránea (relacionándolo con el ID de la tabla Personas). De esta manera asociamos cada pedido con una Empleada que está tramitando. En la siguiente sección de la lección vamos a ver más en profundidad cómo funciona este tipo de restricciones.

Restricciones de FOREIGN KEY

Una relación de `FOREIGN KEY` involucra a una segunda tabla "madre" que tiene los valores originales de esa columna. Luego en la tabla que estamos definiendo tiene valores que referencian a la tabla "madre". Las restricciones de `FOREIGN KEY` deben ser definidas en la tabla "hija". La sintaxis que se debe seguir a la hora de definir la restricción (y sus opciones) es la siguiente:

```

FOREIGN KEY (
    nombre_columna,
    ...
) REFERENCES tabla_madre (columna_madre,...)
    [ON DELETE opcion_referencia]
    [ON UPDATE opcion_referencia]

```

`ON DELETE` indica las acciones a seguir cuando se elimine un registro en la tabla "madre", mientras que `ON UPDATE` indica que acciones tomar cuando se actualice un registro en la tabla "madre".

Algunos valores posibles para la opción_referencia son: RESTRICT, CASCADE y SET NULL. Otros SGBD pueden tener opciones adicionales, pero estas tres son comunes a todos.

- RESTRICT : es similar a omitir la inclusión de ON DELETE u ON UPDATE . Lo que hace esta opción es rechazar el borrado o la actualización de la columna clave en la tabla "madre".
- CASCADE : borrar o actualizar una fila/registro en la tabla "madre" hace que las filas correspondiente de la tabla "hija" se borren o se actualicen en consecuencia.
- SET NULL : borrar o actualizar una fila/registro en la tabla "madre" hace que la columna correspondiente de la tabla "hija" se actualice al valor NULL para los registros afectados.

Creando tablas refenciadas con claves foraneas

Vamos ahora a explicar brevemente como establecemos relaciones entre diferentes tablas cuando las estamos creando.

Supongamos que tenemos una tabla A y una tabla B,

La sintaxis general considerando que existe la tabla A a la cual queremos hacer referencia y que es considerada la tabla madre, que tiene una clave primaria asignada. Para la tabla B, que será la tabla hija:

```
CREATE TABLE IF NOT EXISTS `tabla_B` (
  `id_tabla_B` INT NOT NULL AUTO_INCREMENT,
  `id_tabla_A` INT NOT NULL,
  PRIMARY KEY (`id_tabla_B`),
  CONSTRAINT `fk_tablaB_tablaA`
    FOREIGN KEY (`id_tabla_A`)
    REFERENCES `tabla_A` (id_tabla_A)
    [ON DELETE opcion_referencia]
    [ON UPDATE opcion_referencia])
ENGINE = InnoDB;
```

Vamos a ilustrarlo con un ejemplo:

Tenemos la tabla *alumnas* y la tabla de notas de las alumnas, queremos crear la tabla de notas referenciando a los valores de la tabla de alumnas.

La tabla Alumnas:

id_alumna	nombre	apellido
1	Ana	González
2	Maria	López
3	Lucía	Ramos
4	Elena	Bueno
5	Rocío	García

La tabla de notas de las alumnas :

id_notas	id_alumna	nota
1	1	5
2	2	8
3	3	9
4	4	10
5	5	7

Para crear la tabla de notas referenciada a la tabla de alumnas deberíamos utilizar el siguiente código:

Para la creación de la tabla alumnas:

```
CREATE TABLE IF NOT EXISTS `alumnas` (
  `id_alumna` INT NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(45),
  `apellido` VARCHAR(45),
  PRIMARY KEY (`id_alumna`)
)
ENGINE = InnoDB;
```

Para la creación de la tabla de notas, referenciando los valores de la tabla de alumnas:

```
CREATE TABLE IF NOT EXISTS `notas_alumnas` (
  `id_notas` INT NOT NULL AUTO_INCREMENT,
  `id_alumna` INT NOT NULL,
  `nota` INT NOT NULL,
  PRIMARY KEY (`id_notas`),
  CONSTRAINT `fk_notas_alumnas_alumnas`
    FOREIGN KEY (`id_alumna`)
    REFERENCES `alumnas` (id_alumna) ON DELETE CASCADE ON UPDATE CASCADE
)
ENGINE = InnoDB;
```

Ejercicio: Crea las tabla Empleadas y empleadas_en_proyectos que hemos usado en lecciones anteriores, definiendo claves primarias, claves foránea, tipos de datos, etc. Haz que cuando se elimine una Empleada, se eliminen todas las entradas en empleadas_en_proyectos asociadas.

Ejemplo de Empleadas:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Ejemplo de empleadas_en_proyectos:

id_empleada	id_proyecto
1	1
1	2
2	1
3	2
3	3
3	5
4	2
5	3

Crear una tabla usando una tabla existente

Se puede crear una tabla como una copia de otra existente usando `CREATE TABLE`. Veámoslo con un ejemplo:

```
CREATE TABLE nueva_tabla AS  
  
SELECT nombre_columna1,  
       nombre_columna2  
FROM tabla_original;
```

La nueva tabla se ha creado como una selección de las columnas de una tabla existente. Pueden seleccionarse todas las columnas o sólo un conjunto de ellas. En este caso además, la nueva tabla creada contendrá los mismos valores que la tabla original. Adicionalmente pueden incluirse condiciones con la cláusula `WHERE` para seleccionar qué valores de la tabla original se quieren incluir y cuáles no.

Ejercicio: Crea una copia de la tabla Pedidos que definimos anteriormente, pero sin incluir el NumeroPedido dentro de la misma. Haz que se llame PedidosPorPersona.

ENUNCIADO EJERCICIOS

EJERCICIO 0

Antes de hacer nada, crea un nuevo esquema y actívalo, para no tener conflictos. Utiliza los siguientes comandos:

```
CREATE SCHEMA creacion_tienda; USE creacion_tienda;
```

EJERCICIO 1

Crea la tabla customers (sin tener en cuenta claves foráneas relacionadas con la tabla Employees).

EJERCICIO 2

Crea la tabla employees (sin tener en cuenta claves foráneas relacionadas con la tabla Customers).

EJERCICIO 3

Crea de nuevo las dos tablas(con un nombre diferente a las dos creadas anteriormente) teniendo en cuenta las claves foráneas y restricciones para los datos (por ejemplo que el teléfono tenga sólo 9 cifras, que el límite de crédito nunca sea negativo... Todo lo que se te ocurra).

Repaso creación de tablas

- CREATE TABLE nombre_tabla () para crear una tabla. Siempre será necesario definir lo siguiente:
 - nombre_columnas
 - tipo_dato (longitud maxima)
 - restricciones (en caso de que haya alguna)
- Tipos de restricciones:
 - Columna:
 - NOT NULL: Evita la existencia de nulos en nuestra columna.
 - PRIMARY KEY: Para definir la clave primaria de nuestra tabla.
 - UNIQUE: No permite valores duplicados
 - CONSTRAINT: Establecer un nombre para las restricciones.
 - REFERENCES: Para definir las referencias a las claves de foráneas de nuestra tabla madre.
 - CHECK: Hace una comprobación de las condiciones impuestas.
 - DEFAULT: Para establecer un valor por defecto: NULL, 0, 33
 - Tabla:
 - PRIMARY KEY: Definimos la clave primaria de la tabla.
 - CASCADE
 - FOREING KEY: Para establecer la clave foránea de otra tabla
 - UNIQUE: No permite duplicados en ninguna de las columnas de la tabla.
 - CHECK: Hace una comprobación de las condiciones impuestas.
 - Restricciones sobre las claves foráneas:
 - RESTRICT: Evitar que los cambios que se apliquen en la tabla que contiene la clave foránea, no afecten en la tabla madre donde dicha clave es primaria
 - CASCADE: Los cambios que se hace sobre la columna clave foránea, también afecte a la tabla donde dicha clave foránea es clave primaria.
 - SET NULL: Los cambios en la tabla donde esta la clave foránea, se conviertan en NULL en la tabla donde es clave foránea
 - Crear una tabla nueva, utilizando otra tabla

```
CREATE TABLE nombre_tabla SELECT ( columnas )
FROM tabla_origen;
```

Pair programming Creación de tablas

Recordad que no es necesario que hagas todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

En esta sesión crearemos una nueva BBDD desde 0 ! Supongamos que tenemos una tienda de zapatillas y que durante mucho tiempo hemos ido recopilando mucha información sobre nuestros empleados, las zapatillas que tenemos, los clientes y todas las facturas que hemos emitido. Nuestra información puede que este un poco desordenado y empieza a ser un poco complicado gestionar toda esa información, por lo que hemos decidido crearnos una BBDD en SQL. Para ello crearemos 4 tablas en la BBDD: - Empleados - Clientes - Facturas - Zapatillas

La tabla Facturas tiene una relación con la tabla Empleados y la tabla Clientes y la tabla Zapatillas. Estas tres últimas no tienen ninguna relación entre ellas.

Antes de nada para poder empezar este ejercicio deberemos crear la base de datos, para ello deberemos escribir:

```
CREATE SCHEMA `tienda_zapatillas`;
```

```
USE `tienda_zapatillas`;
```

De esta forma habremos creado la base de datos que necesitamos y la habremos seleccionado, antes de poder empezar a crear las tablas.

Carácteristicas de nuestras tablas

- Tabla Zapatillas: tiene 3 columnas: id_zapatilla, modelo, color con las siguientes características:
 - `id_zapatilla` : es una clave primaria de tipo int, autoincremental y no nula.
 - `modelo` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `color` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
- Tabla Clientes: tiene 9 columnas id_cliente, nombre, numero_telefono, email, direccion, ciudad, provincia, pais, codigo_postal con las siguientes características:
 - `id_cliente` : es una clave primaria de tipo int, autoincremental y no nula.
 - `nombre` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `numero_telefono` : es *integer* de longitud máxima de 9 caracteres, no nula.
 - `email` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `direccion` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `ciudad` : es una cadena de caracteres de longitud máxima de 45 caracteres, puede ser nula.
 - `provincia` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `pais` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `codigo_postal` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
- Tabla Empleados: tiene 5 columnas id_empleado, nombre, tienda, salario, fecha_incorporacion con las siguientes características:
 - `id_empleado` : es una clave primaria de tipo int, autoincremental y no nula.
 - `nombre` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `tienda` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `salario` : es int, puede ser nula.
 - `fecha_incorporacion` : es una columna de tipo date, no nula.
- Tabla Facturas: tiene 6 columnas id_factura ,numero de factura, fecha, id_empleado, id_cliente, id_zapatilla, con las siguientes características:
 - `id_factura` : es una clave primaria de tipo int, autoincremental y no nula.
 - `numero_factura` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
 - `fecha` : es una columna de tipo date, no nula.
 - `id_zapatilla` : es una clave foránea de tipo int, no nula
 - `id_empleado` : es una clave foránea de tipo int, no nula.
 - `id_cliente` : es una clave foránea de tipo int, no nula.

NOTA En esta última tabla tendremos que incluir todos los `CONSTRAINT` necesarios para establecer los relaciones entre las tablas, donde tendremos que especificar:

- Foreign Key
- References

Alteración de tablas

Consultas de modificación 1

En la anterior lección hemos aprendido a cómo crear tablas en una base de datos. El siguiente paso va a consistir en modificar o incluso eliminar las tablas una vez ya han sido creadas.

ALTER TABLE

El operador `ALTER TABLE` se puede usar para modificar la estructura de una tabla. Por ejemplo, usándolo podríamos añadir o eliminar columnas en una tabla existente. También podríamos modificar las columnas existentes, cambiar su tipo de dato, su nombre o sus restricciones.

Uso general de ALTER TABLE

De cara a poder utilizar `ALTER TABLE` en una tabla, se necesitan tener permisos para realizar operaciones `ALTER`, `CREATE` e `INSERT` en dicha tabla. Renombrar una tabla requiere permisos para hacer `ALTER` y `DROP` en la misma.

Posteriormente al operador `ALTER TABLE` debe indicarse el nombre de la tabla que quiere alterarse. Si no se indicase el nombre de la tabla, el resto de la instrucción no tendría ningún efecto al no saber el objetivo de la misma.

La sintaxis de la mayoría de las alteraciones es similar a la sintaxis empleada en `CREATE TABLE`. Se pueden utilizar los operadores `ADD`, `DROP`, `CHANGE`, etc., (los veremos a continuación en esta lección) seguidos de los nombres de las columnas sobre los que se van a aplicar. Después, los modificadores que se pueden emplear sobre cada una de las columnas siguen también la misma sintaxis que en el caso de `CREATE TABLE`.

Cuando usamos MySQL como SGBD una misma sentencia `ALTER` se pueden incluir múltiples operadores `ADD`, `DROP`, `CHANGE`, etc., cada uno de ellos separado por una coma. Esto es en realidad una extensión sobre el estándar SQL, por lo que cuando uséis otros sistemas de gestión de bases de datos puede no ser posible.

Añadir columnas a una tabla existente

Para añadir columnas a una tabla ya existente podemos utilizar el operador `ADD`:

```
ALTER TABLE nombre_tabla  
ADD COLUMN nombre_columna tipo_de_datos restricciones;
```

La palabra `COLUMN` es opcional y se podría omitir, excepto en el caso en el que queramos renombrar una columna ya existente (`RENAME COLUMN`), ya que en ese supuesto sería necesaria para diferenciar la operación de aquella en la que queramos renombrar una tabla entera (`RENAME`).

Los tipos de datos y las restricciones que se pueden aplicar son las mismas que hemos usado cuando creábamos las tablas con `CREATE TABLE` (podéis encontrarlas en la lección anterior sobre `CREATE TABLE`).

Ejemplo: Añade a la tabla "empleadas" una columna llamada "salario" que contenga un número con decimales.

Considerando que partimos de la tabla empleadas que contiene los siguientes campos:

```
ALTER TABLE empleadas  
ADD COLUMN salario FLOAT ;
```

Eliminar columnas de una tabla existente

Para eliminar columnas ya existentes en una tabla, seguiremos la siguiente estructura:

```
ALTER TABLE nombre_tabla  
DROP COLUMN nombre_columna;
```

En este caso la palabra `COLUMN` también es opcional. `DROP COLUMN` también borrará de la base de datos todos los datos existentes en dicha columna, así que se debe utilizar con mucho cuidado.

También estaría la opción (sólo en MySQL) de eliminar múltiples columnas en una sola sentencia `ALTER`:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

De usar algún otro SGBD que no fuese compatible con esta funcionalidad, habría que escribir una sentencia `ALTER` independiente por cada columna que se quiera eliminar de la tabla.

Cambiar el tipo de datos de una columna

Otra opción interesante que nos proporciona `ALTER TABLE` es la de modificar el tipo de datos de una columna de la tabla. La sintaxis a seguir sería la siguiente:

```
ALTER TABLE nombre_tabla  
MODIFY COLUMN nombre_columna tipo_de_datos;
```

Una vez más la palabra `COLUMN` es opcional. Los tipos de datos que podemos utilizar aquí son los mismos que se pueden usar con `CREATE TABLE` (los hemos descrito en la lección correspondiente).

Cambiar las restricciones de una tabla/columna

También podría interesarnos cambiar las restricciones de alguna de las columnas. Dichas restricciones se definen en la consulta `CREATE TABLE`. Algunas de ellas son `PRIMARY KEY`, `UNIQUE`, `REFERENCE`, `NOT NULL`, etc.

NOTA: No se permite modificar una restricción o `CONSTRAINT` ya existente. El procedimiento a realizar sería eliminar primero la `CONSTRAINT` existente con `DROP CONSTRAINT`:

```
ALTER TABLE nombre_tabla  
DROP CONSTRAINT nombre_restriccion;
```

Después se podría añadir una nueva restricción usando `ADD CONSTRAINT`. Podemos usar `ADD CONSTRAINT` con los operadores `CHECK`, `FOREIGN KEY`, `PRIMARY KEY`, `UNIQUE`. Para añadir por ejemplo una `FOREIGN KEY` haríamos:

```
ALTER TABLE nombre_tabla  
ADD CONSTRAINT nombre_restriccion  
    FOREIGN KEY (nombre_columna)  
    REFERENCES tabla_madre (columna_referencia)  
ON UPDATE CASCADE;
```

En el ejemplo anterior hemos utilizado la opción `ON UPDATE CASCADE`. Esta es una opción que se puede usar cuando se indican las `FOREIGN KEY` en una tabla (la definimos en la lección de `CREATE TABLE`).

Una última posibilidad consiste en añadir restricciones a la tabla entera, que pueden involucrar a más de una columna. En el siguiente ejemplo podemos observar cómo se definiría la clave primaria de la tabla como una combinación de columnas:

```
ALTER TABLE Table  
ADD CONSTRAINT  
    PRIMARY KEY(Columna1,Columna2,...);
```

Ejercicio: Cambia el tipo de datos de la columna "salario" de la tabla "empleadas" de tipo float a un número entero.

Ejercicio: Sobre la tabla empleadas anterior añade la restricción de que el valor de salario no pueda estar vacío para nuevos registros.

Ejemplo: Añade una restricción o restricciones a la columna "DNI" de la tabla "Empleadas2" para que sea un atributo que no pueda repetirse ni dejarse vacío al introducir un nuevo registro en la base de datos.

Considerando que partimos de la tabla empleadas que contiene los siguientes campos:

Primero añadimos la columna DNI.

```
ALTER TABLE empleada2  
ADD COLUMN dni VARCHAR(30);
```

Opción A: Convertirlo en clave primaria

```
ALTER TABLE empleada2  
ADD CONSTRAINT  
  
    PRIMARY KEY (dni);
```

Opción B: Imponer restricciones `UNIQUE` y `NOT NULL`:

```
ALTER TABLE empleada2  
ADD CONSTRAINT  
    UNIQUE (dni);  
  
ALTER TABLE Empleada2  
MODIFY dni CHAR NOT NULL;
```

DROP TABLE

La sentencia `DROP TABLE` nos permite eliminar permanentemente una tabla de la base de datos. Esta sentencia elimina tanto la estructura de la tabla como todos los datos almacenados en la misma. La sintaxis que debemos seguir para utilizarla correctamente es:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nombre_tabla [,nombre_tabla2,...] [CASCADE];
```

Vamos a ver en detalle cada una de las opciones y operadores que componen la sentencia:

- `DROP TABLE` indica que queremos borrar la tabla y todos sus registros.
- `TEMPORARY` es un operador opcional que indica que solo queremos borrar las tablas temporales (más adelante veremos que son estas tablas).
- La opción `IF EXISTS` hace que solo se elimine la tabla si esta existe (mostrará un warning por pantalla si la tabla no existe). No es obligatorio ponerlo.
- `CASCADE` hace que cada vez que se elimine un registro de la tabla, todas las filas relacionadas en otras tablas (debido a claves foráneas por ejemplo) también se eliminan.

Ejemplo: Usa `DROP` para eliminar una tabla llamada `tabla1`:

```
DROP TABLE tabla1;
```

Ejemplo: Usa `DROP` para eliminar la `tabla1` sólo si esta existe:

```
DROP TABLE IF EXISTS tabla1;
```

Ejemplo: Elimina tres tablas (`tabla1`, `tabla2` y `tabla3`) usando una única sentencia `DROP`:

```
DROP TABLE IF EXISTS tabla1, tabla2, tabla3;
```

Tablas temporales

Las tablas temporales se usan para almacenar conjuntos de datos durante un periodo corto de tiempo. La idea es usar esos datos brevemente y luego borrar la tabla. Por ejemplo, podríamos haber ejecutado una consulta `SELECT` compleja que contuviese varios `JOIN`s, etc., lo cual haría que su ejecución fuese lenta. Almacenar el resultado en una tabla temporal es interesante para poder ahorrarnos ese tiempo de ejecución si necesitamos acceder a los datos varias veces. Después de usarlos, borraríamos dicha tabla con la siguiente sentencia `DROP`:

```
DROP TEMPORARY TABLE IF EXISTS tabla1;
```

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla que hemos creado usando la siguiente sentencia SQL:

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

La tabla `t1` tiene las columnas `a`, que consiste en un número entero, y la columna `b`, que consiste en una cadena de caracteres de 10 elementos.

EJERCICIO 1

Renombra la tabla `t1` a `t2`.

EJERCICIO 2

Cambia la columna `a` de tipo `INTEGER` a tipo `TINYINT NOT NULL` (manteniendo el mismo nombre para la columna).

EJERCICIO 3

Cambia la columna `b` de tipo `CHAR` de 10 caracteres a `CHAR` de 20 caracteres. Ya que estamos renombrando la columna como `c`.

EJERCICIO 4

Añade una nueva columna llamada `d` de tipo `TIMESTAMP`.

EJERCICIO 5

Añade una columna `INDEX` llamada `d` y una cláusula `UNIQUE` a la columna `a`.

EJERCICIO 6

Elimina la columna `c` que definiste en el ejercicio 3.

EJERCICIO 7

Crea una tabla llamada t3 idéntica a la tabla t2 (de manera automática, no definiéndola columna a columna).

EJERCICIO 8

Elimina la tabla original t2 y en otra sentencia renombra la tabla t3 como t1.

Repaso Modificación de tablas

- Para modificar tablas utilizamos : ALTER TABLE nombre_tabla
Y nos permite añadir, eliminar , modificar datos, columnas y restricciones.
- ADD COLUMN nombre_columna tipo_dato restriccion(opcional): Añadimos nuevas columnas

```
ALTER TABLE nombre_tabla  
ADD COLUMN nombre_columna;
```

- RENAME COLUMN nombre_columna: Para cambiar el nombre de una columna.

```
ALTER TABLE nombre_tabla  
RENAME COLUMN nombre_columna;
```

- DROP COLUMN nombre_columna: Para eliminar una columna.

```
ALTER TABLE nombre_tabla  
DROP COLUMN nombre_columna;
```

- MODIFY COLUMN nombre_columna tipo_dato_nuevo: Cambiar los tipos de datos o las restricciones.

```
ALTER TABLE nombre_tabla  
MODIFY COLUMN nombre_columna tipo_dato_nuevo;
```

- Para modificar columnas con restricciones o para añadir nuevas restricciones:
 - ADD CONSTRAINT nombre_restriccion: Para añadir una restriccion
 - DROP CONSTRAINT nombre_restriccion: Para eliminar una restriccion. Sera obligatorio para poder modificar columnas que tengan ya restricciones previas, implementado como método de seguridad.
- DROP TABLE nombre_tabla: Eliminar una tabla de nuestra base de datos , de forma permanente. A no ser que tengamos una copia previa de la base datos.

```
DROP TABLE nombre_tabla;
```

- Operadores adicionales:
 - CASCADE : Cada vez que eliminamos un registro de la tabla principal, todas las tablas que tengan tambien referencia a esta serán eliminadas
 - TEMPORARY : Para eliminar únicamente las tablas temporales
 - IF EXIST : Comprobar previamente si la tabla existe antes de probar a eliminarla.

Pair programming alteración de tablas

Recordad que no es necesario que hagáis todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

Seguimos trabajando con la BBDD que creamos en la sesión de *pair programming* anterior. Revisando nuestras tablas nos hemos dado cuenta que algunas tienen algunos errores. En algunas tablas nos faltan columnas, en otras hemos introducido columnas de más o incluso nos hemos equivocado a la hora de especificar el tipo de los datos.

Actividades

En este ejercicio vamos a corregir los errores que hemos encontrado en nuestras tablas.

- Tabla Zapatillas:

Se nos ha olvidado introducir la marca y la talla de las zapatillas que tenemos en nuestra BBDD. Por lo tanto, debemos incluir:

- `marca` : es una cadena de caracteres de longitud máxima de 45 caracteres, no nula.
- `talla` : es un entero, no nulo.

- Tabla Empleados

- `salario` : es un entero, no nulo. Pero puede que el salario de nuestros empleados tenga decimales, por lo que le cambiaremos el tipo a decimal.

- Tabla Clientes

- `pais` : la hemos incluido en la tabla pero nuestro negocio solo distribuye a España, por lo que es una columna que no hará falta. La eliminaremos.
- `codigo_postal` : es un *string*, pero esto no tiene mucho ya que son números de longitud fija de 5 caracteres. Por lo tanto, cambiaremos el tipo a entero de longitud 5.

- Tabla Facturas:

- `total` : madre mía!!! Se nos ha olvidado incluir el total de la cada factura generada !Creemos esa columna con un tipo de datos decimal.

Happy coding

Inserción de datos

Consultas de modificación 2

Ahora que ya tenemos soltura con algunas de las instrucciones de SQL para extraer algunos datos, vamos a proceder a ver que es lo que debemos hacer cuando queremos realizar alguna modificación sobre los datos contenidos en la base de datos con la que estamos trabajando. Ya sea porque hemos encontrado valores mal introducidos o erróneos, se nos ha proporcionado datos nuevos y han de ser introducidos en la base de datos o se nos ha pedido eliminar algún registro existente.

INSERT

La instrucción `INSERT` se utiliza para introducir nuevos datos en una base de datos. Para ello es de vital importancia resaltar que se ha de especificar la tabla concreta en la que se quiera realizar la inserción de nuevos datos.

```
INSERT INTO nombre_tabla (columna1, columna2, columna3, columna4, ...)
VALUES (valor1, valor2, valor3, valor4, ....);
```

Para hacer uso de esta instrucción se han de tener en cuenta un par de consideraciones previas a su uso y de importancia para poder ejecutarla correctamente.

1. Sobre qué tabla queremos realizar la inserción de nuevos datos.
2. Las columnas que posee la tabla seleccionada. (La variable de ID, actúa como clave primaria, no es necesario considerarla.)
3. Los tipos de datos contenido en cada columna de la tabla escogida. Esto se debe a que se ha de tener en cuenta si estamos trabajando con texto, números o fechas.
 - Los textos deben ser insertados entre comillas simples ' '. Los números de teléfono se recomienda introducirlo como texto, ya que pueden incluir 0 delante o el prefijo que le corresponda.
 - Los valores numéricos, los decimales siempre se indican con el separador . , en lugar de con la coma como estamos acostumbrados. Aquellos números que empiecen por 0, se descartará el 0 inicial, debido a que MySQL considera que no es relevante.
 - Las fechas deben siempre introducirse en el formato AAAA-MM-DD (AÑO-MES-DIA), [notación ISO 8601](#).

De nuevo consideraremos la tabla Empleadas perteneciente a una compañía:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia

Si ahora se nos pidiera introducir un nuevo registro en la tabla deberíamos utilizar la instrucción `INSERT`, identificamos primero las consideraciones explicadas anteriormente antes de hacer uso de nueva la sentencia.

- La tabla sobre la que está trabajando es Empleadas
- La tabla contiene las columnas: salario, nombre, apellido, email, telefono, ciudad, pais.
- Las columnas tienen los siguientes tipos de datos: Numérico, Texto, Texto, Texto, Texto, Texto, Texto.

Una vez conocido esto, procedemos a realizar la inserción de los datos de la siguiente forma:

```
INSERT INTO empleadas (salario, nombre, apellido, email, telefono, ciudad, pais)
VALUES (2000, 'Inés', 'Romero', 'ines@adalab.es', '619739261', 'Sevilla', 'España');
```

Tras ejecutar la sentencia anterior, la tabla se habrá actualizado y quedará de la siguiente forma:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España

Como podemos ver se habrá añadido un registro adicional, al final del último registro que existía previamente a la inserción. En el caso de que tuviéramos un valor a introducir y no conociéramos todos los campos requeridos, tampoco sería un problema. Dichos campos desconocidos se completarían con un `null`, sería un valor faltante, siempre que al crear la tabla se hubiera especificado la restriccion en las columnas que `DEFAULT NULL` para el caso de valores desconocidos.

Nos piden introducir los datos de la siguiente empleada y se han olvidado de proporcionarnos los datos de email y ciudad. Podemos ver como quedaría la tabla Empleadas al insertar la nueva empleada en la tabla:

```
INSERT INTO empleadas (salario, nombre, apellido, email, telefono, ciudad, pais)
VALUES (2200, 'Alba', 'Fernández', 'alba@adalab.es', 'Portugal');
```

Tras correr la sentencia anterior, la tabla se habrá actualizado y quedará de la siguiente forma:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España

Así mismo, en caso de necesitar introducir más de un registro nuevo, se puede utilizar la sentencia `INSERT`, para realizarlo todo en una misma instancia, en lugar de tener que ir una por una. Para ello, lo ilustramos en el siguiente ejemplo.

Introduciremos ahora 3 nuevas empleadas a la tabla existente, para hacerlo es necesario poner una coma a continuación del siguiente registro a introducir después de los datos de cada nueva empleada:

```
INSERT INTO empleadas (salario, nombre, apellido, email, telefono, ciudad, pais)
VALUES (1800, 'Julia', 'Aguilar', 'julia@adalab.es', '614339261', 'Zaragoza', 'España'),
(2000, 'Irene', 'Montenegro', 'irene@adalab.es', '659745615', 'Cataluña', 'España'),
(3000, 'Laura', 'Navarro', 'laura@adalab.es', NULL,NULL, 'Italia'), ;
```

La tabla resultante quedaría de las siguiente forma:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España
8	1800	Julia	Aguilar	julia@adalab.es	614339261	Zaragoza	España
9	2000	Irene	Montenegro	irene@adalab.es	659745615	Cataluña	España
10	3000	Laura	Navarro	laura@adalab.es	null	null	Italia

UPDATE

La cláusula `UPDATE` se utiliza para actualizar los datos existentes en algunos de los registros ya existentes de una tabla. Por ejemplo, en el caso de que se nos haya proporcionado información actualizada o se nos haya compartido algún campo que estuviera faltante para alguno de los registros.

De esta forma para utilizar esta instrucción debemos seguir la siguiente sintaxis:

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2, ....
WHERE condición= valor_condición;
```

NOTA: Hay que ser extremadamente cauteloso al actualizar registros en una tabla. Es muy importante fijarse en la cláusula `WHERE` en la instrucción `UPDATE`. `WHERE` indica los registros que deben ser actualizados. Si se omite esta cláusula se actualizarán **TODOS LOS REGISTROS DE LA TABLA!**

Cosa que no queremos hacer nunca de los jamases, salvo que quieras ver el mundo arder



This is fine.

Vamos a ver un caso práctico haciendo uso de nuevo de la tabla de empleadas, tal y como la tenemos con los últimos registros introducidos en el apartado anterior. Imaginemos ahora que la última empleada (Laura) se ha mudado a Madrid, en España, y nos ha proporcionado un número de teléfono. Para introducir esta información haríamos uso de la instrucción `INSERT`, utilizando como condición en el `WHERE` el `IDEmpleada`, para únicamente actualizar ese registro. En los casos en los que se vaya a actualizar únicamente una de las filas de la tabla, se recomienda utilizar como condición el índice de la tabla asignado a dicho registro, para evitar actualizar más de un registro.

```
UPDATE empleadas  
SET telefono = 678652840, ciudad = Madrid, pais = España  
WHERE id_empleada = 10;
```

Tras ejecutar la sentencia anterior, la tabla de empleadas nos quedaría de la siguiente manera:

<code>id_empleada</code>	<code>salario</code>	<code>nombre</code>	<code>apellido</code>	<code>email</code>	<code>telefono</code>	<code>ciudad</code>	<code>pais</code>
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España
8	1800	Julia	Aguilar	julia@adalab.es	614339261	Zaragoza	España
9	2000	Irene	Montenegro	irene@adalab.es	659745615	Cataluña	España
10	3000	Laura	Navarro	laura@adalab.es	678652840	Madrid	España

Si por algún error o por simple malicia, nos olvidásemos de la condición de `WHERE`, habría ocurrido lo siguiente:

```
UPDATE empleadas
```

```
SET telefono = 678652840, ciudad = Madrid, pais = España
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	678652840	Madrid	España
2	4000	Maria	López	maria@adalab.es	678652840	Madrid	España
3	3000	Lucía	Ramos	lucia@adalab.es	678652840	Madrid	España
4	5000	Elena	Bueno	elena@adalab.es	678652840	Madrid	España
5	1500	Rocío	García	rocio@adalab.es	678652840	Madrid	España
6	2000	Inés	Romero	ines@adalab.es	678652840	Madrid	España
7	2200	Alba	Fernández	alba@adalab.es	678652840	Madrid	España
8	1800	Julia	Aguilar	julia@adalab.es	678652840	Madrid	España
9	2000	Irene	Montenegro	irene@adalab.es	678652840	Madrid	España
10	3000	Laura	Navarro	laura@adalab.es	678652840	Madrid	España

En caso de que esto ocurriera, la única forma de solucionarlo sería utilizar una copia de seguridad anterior, para restaurar el estado de la tabla antes de la última instrucción de actualización de datos.

Imaginemos que hemos restaurado a la versión anterior y hemos recuperado la tabla donde los datos de Telefono y ciudad no son el mismo para todos los registros.

NOTA: Por defecto MySQL workbench viene con el modo seguro activado, lo que impide ejecutar secuencias de actualización o borrado de datos sin la condición `WHERE`, para evitar problemas actualizando datos.

De forma opcional podemos incluir los siguientes campos, si por ejemplo queremos actualizar varias filas utilizando una condición común para una serie de filas de la tabla:

```
UPDATE nombre_tabla  
SET columna1 = valor1, columna2 = valor2, ....  
WHERE condiciones= valor_condiciones  
[ORDER BY ...] (argumento opcional)  
[LIMIT row_count] (argumento opcional);
```

Imaginemos ahora que han subido el salario mínimo interprofesional a 2000 euros. De esta forma tendríamos que actualizar algunos de los sueldos de la tabla de Empleadas. Podríamos hacer esto utilizando como condición el valor de la columna salarios.

Realizar lo anterior haciendo uso de la siguiente instrucción:

```
UPDATE empleadas  
SET salario = 2000  
WHERE salario <2000;
```

De esta forma la tabla anterior nos cambiaría 2 de los registros de la tabla, y nos quedaría de la siguiente forma:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	2000	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España
8	2000	Julia	Aguilar	julia@adalab.es	614339261	Zaragoza	España
9	2000	Irene	Montenegro	irene@adalab.es	659745615	Cataluña	España
10	3000	Laura	Navarro	laura@adalab.es	678652840	Madrid	España

NOTA: Es posible que con el modo seguro, tampoco permita actualizar más de un registro, basado en una condición. Más vale ser precavido, que acabar rompiendo las tablas!

DELETE

La instrucción `DELETE` se utiliza de forma similar al explicado en el apartado anterior. En este caso su ejecución elimina una o varias de las filas de la tabla. Esto es útil cuando deseamos quitar registros de una tabla. La sintaxis a emplear para hacer uso de esta instrucción es la siguiente:

```
DELETE FROM empleadas
WHERE condición = valor_condición
```

De forma opcional podemos incluir los siguientes campos, si por ejemplo queremos actualizar varias filas utilizando una condición común para una serie de filas de la tabla:

```
DELETE FROM nombre_tabla
WHERE condiciones= valor_condiciones
[ORDER BY ...] (argumento opcional)
[LIMIT row_count] (argumento opcional);
```

NOTA: Hay que ser extremadamente cauteloso al actualizar registros en una tabla. Es muy importante fijarse en la cláusula WHERE en la instrucción DELETE . WHERE indica los registros que deben ser actualizados. Si se omite esta cláusula se actualizaran **TODOS LOS REGISTROS DE LA TABLA!**

Vamos a ver ahora un ejemplo sencillo del uso de esta nueva instrucción haciendo uso de nuevo de nuestra tabla de Empleadas. Para ello imaginemos que una de las empleadas ha dejado la empresa y nos han pedido que eliminemos su registro de la base de datos. Concretamente tenemos que eliminar los datos pertenecientes a Julia Aguilar. De esta forma tendríamos que ejecutar la siguiente instrucción.

```
DELETE FROM empleadas
WHERE id_empleada = 8
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	2000	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España
9	2000	Irene	Montenegro	irene@adalab.es	659745615	Cataluña	España
10	3000	Laura	Navarro	laura@adalab.es	678652840	Madrid	España

Alternativamente podríamos haber eliminado un campo sin hacer uso del IDEmployed, utilizando más de una condición para conseguir el mismo resultado:

```
DELETE FROM empleadas  
WHERE nombre = 'Julia' AND apellido= 'Aguilar';
```

Vamos ahora a ver como podríamos eliminar más de una fila haciendo uso de una única condición. Por ejemplo supongamos que ahora se ha puesto un tope al salario máximo de las empleadas y todas menos una de las empleadas que cobran más de 2500 euros han decidido dejar la empresa. De esta forma necesitaremos eliminar diversos registros. Por tanto podemos hacer uso de la siguiente instrucción para hacerlo de una vez.

```
DELETE FROM empleadas  
WHERE salario > 2500  
ORDER BY salario DESC  
LIMIT 3;
```

Al ejecutar la sentencia la tabla anterior nos quedaría de la siguiente forma:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@ad.alab.es	654785214	Madrid	España
3	3000	Lucía	Ramos	lucia@ad.alab.es	674459123	Valencia	España
5	2000	Rocío	García	rocio@ad.alab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@ad.alab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@ad.alab.es	null	null	España
9	2000	Irene	Montenegro	irene@ad.alab.es	659745615	Cataluña	España

Se ha empleado el salario como forma de ordenar la tabla, ya que teníamos 4 empleadas que cobraban más de 2500 euros, pero una de ellas ha decidido quedarse en la empresa.

Finalmente vamos a ver que pasaría, si nos olvidásemos de incluir la condición a la hora de realizar la eliminación de registros. Esto causaría que se eliminaran todos los registros de la tabla de empleadas, sin eliminar la tabla en sí.

```
DELETE FROM empleadas;
```

NOTA: De nuevo el modo seguro de MySQL workbench, nos impide ejecutar esta sentencia, ya que en los casos habituales de uso **NUNCA** desearemos dejar una tabla vacía.

La tabla se habría quedado completamente vacía. De ahí la precaución que se ha de tomar a la hora de eliminar valores de las tablas, para no terminar cargándose tablas enteras por error.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar las tablas ya creadas llamadas `customers` (clientes/as) y `employees`, que están en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisa la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

EJERCICIO 0

Piensa en la información que puede contener cada columna y el tipo de datos que puede contener. Incluye esta información en forma de comentario, sobre tu script de sql.

Recordatorio: Los comentarios se introducen utilizando `/* texto del comentario */`

EJERCICIO 1

Crea una copia de la tabla Customers, ya que vamos a modificar los datos originales de dicha tabla. Para ello ejecuta la siguiente sentencia:

```
CREATE TABLE IF NOT EXISTS customers_mod
SELECT *
FROM customers;
```

Así mismo vamos a desactivar el modo seguro para poder realizar los ejercicios posteriores. Para ello: Pestaña Editar -> Preferencias -> Editor SQL -> Deseleccionar la opción de Actualizaciones segura (rechaza `UPDATE`s y `DELETE`s sin restricciones) -> Desconéctate del servidor y vuelve a conectarte o alternativamente cierra MySQL y vuelve a abrirlo.

EJERCICIO 2

Realiza una inserción de datos sobre la tabla `customers` introduciendo la siguiente información.

- customernumber: 343
- customername: Adalab
- contactlastname: Rodriguez
- contactfirstname: Julia
- phone: 672986373
- addressline1: Calle Falsa 123
- addressline2: Puerta 42
- city: Madrid
- state: España
- postalcode: 28000
- country: España
- salesrepemployeeNumber: 15
- creditlimit: 20000000

EJERCICIO 3

Realiza una inserción de datos sobre la tabla Customers introduciendo la siguiente información. Fíjate que ahora no tenemos toda la información.

- customernumber: 344
- customername: La pegatina After
- contactlastname: Santiago
- contactfirstname: Maricarmen
- phone: 00000000
- addressline1: Travesía del rave
- addressline2: NULL
- city: Palma de Mallorca
- state: NULL
- postalcode: 07005
- country: España
- salesrepemployeeNumber: 10
- creditlimit: 45673453

EJERCICIO 4

Introduce ahora 5 filas nuevas con la información que consideres relevante para los campos disponibles en una misma instrucción. Se recuerda que el Índice(=la clave primaria), no es necesaria especificarla. En 3 de las nuevas filas debes dejar vacío el campo 'contactFirstName'

EJERCICIO 5

Actualiza ahora los datos faltantes correspondientes al CustomerName 'La pegatina After' con la siguiente información.

- addressline1: Polígono Industrial de Son Castelló
- addressline2: Nave 92
- city: Palma de Mallorca
- state: España
- postalcode:28123
- country: España
- salesrepemployeenumber: 25
- creditlimit: 5000000

EJERCICIO 6

Vamos ahora a romper a propósito la tabla con la que estamos trabajando, para ello primero vamos a realizar una copia de la misma antes de ejecutar lo siguiente. Con el nombre customers_destroy.

Para ello hacemos uso de la herramienta de exportación de datos de MySQL, como se explicaba en las guías del módulo 0 para la exportación y la importación de datos.

Una vez creada la copia y guardada a buen recaudo. Vamos a actualizar algunos de los cambios sin especificar la condición del WHERE . Para ello modifica el campo los siguientes campos de

- addressline1: Vamos
- addressline2: a
- postalcode: fastidiar
- country: la tabla :)

Y observa los frutos de tu metedura de pata.

Tras esto restaura la tabla que has trasteado con la copia que te has creado previamente.

EJERCICIO 7

Actualiza ahora los datos de la tabla customers_mod, para que las 10 primeras empresas sean gestionadas por la representante de ventas numero 2. El resto de empresas no deben ser modificadas.

EJERCICIO 8

Queremos ahora eliminar de los datos de la tabla aquellos registros que contengan un 'null' en el campo 'ContactFirstName'.

EJERCICIO 9

Eliminar ahora de los datos los últimos 5 registros de la tabla ordenando por la columna creditLimit en orden de mayor a menor, queremos eliminar las 5 empresas con menor liquidez.

EJERCICIO 10

Ejecuta la instrucción de `DELETE` para la tabla `customers_mod` olvidando el `WHERE` como condición. Y observa lo que ha ocurrido. (Y medita sobre las decisiones que has tomado en tu vida, si esto ocurriera en la base de datos de la empresa en la que trabajases, como mínimo te caería una bronca de narices.)

EJERCICIO 11

Con el fin de que no se vuelva a repetir el ejecutar la sentencia `DELETE` sin el `WHERE`, mira el siguiente vídeo

[No te olvides de poner el Where en el Delete From.](#)

Repaso

En esta lección hemos complementado los conocimientos básicos de SQL aprendiendo a usar otros operadores e instrucciones más avanzadas:

- `INSERT` : Se utiliza para introducir nuevos valores dentro de la tabla seleccionada, si se deja algún campo vacío del total de columnas, se introduce un null por defecto.
- `UPDATE` : Se utiliza para reescribir los valores existentes de una columna por unos nuevos valores, es importante utilizar el `WHERE` con una condición, para solo actualizar los valores deseados. Si se omite, actualizará todos los valores de dicha columna para la totalidad de la tabla.
- `DELETE` : Se utiliza para eliminar filas de la tabla. Es imprescindible de nuevo utilizar el `WHERE` con una condición, para eliminar únicamente los valores deseados. Si se omite, se eliminará todo el contenido de la tabla.

Repaso inserción datos

- Para introducir nuevos datos en una tabla existente:

```
INSERT INTO nombre_tabla (columnas)
VALUES (valores_columnas)
```

- Se debe de tener en cuenta lo siguiente por tipo de valor:
 - Si intentas introducir valores diferentes al tipo de datos asignado a la columna nos dará un error.
 - Valores numéricos los decimales se separan utilizando un . en lugar de la , como estamos acostumbradas.
 - Texto: Se utilizan comillas simples "
 - Fechas: Se deben de escribir como AAAA-MM-DD.
 - Recomendaciones: Como las columnas numéricas no considera los ceros por delante, deberemos crear las columnas como tipo texto. Vease: Códigos postales, números de teléfono.
- Podemos insertar más de un valor en una misma tabla, deberemos añadir los registros a introducir separados por comas.

```
INSERT INTO nombre_tabla (columnas)
VALUES (valores_columnas), (valores_columnas), (valores_columnas)...;
```

- No es necesario indicar el valor de la clave primaria al introducir nuevos registros.
- Para actualizar valores ya existentes en la tabla:

```
UPDATE nombre_tabla SET columna1= valor1, columna2=valor2 ...
WHERE condicion= valor_condicion;
```

- SIEMPRE debemos añadir el WHERE, ya que si no lo hacemos modificaremos todos los registros de la tabla.
- Para borrar registros de una tabla:

```
DELETE FROM nombre_tabla
WHERE condicion= valor_condicion;
```

- SIEMPRE debemos añadir el WHERE, ya que si no lo hacemos dejaremos la tabla completamente vacía.

Pair programming Inserción de datos

Recordad que no es necesario que hagas todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

Genial!!! Ya tenemos nuestra BBDD creada y con los errores solucionados. Ahora es el momento de manipularla un poco. Vamos a meter datos, actualizar datos que hayamos metido o incluso eliminarlos!!!

Actividades

1. Lo primero que vamos a hacer es insertar datos en nuestra BBDD con los siguientes datos:

- Tabla zapatillas

id_zapatilla	modelo	color	marca	talla
1	XQYUN	Negro	Nike	42
2	UOPMN	Rosas	Nike	39
3	OPNYT	Verdes	Adidas	35

- Tabla empleados

id_empleado	nombre	tienda	salario	fecha_i on
1	Laura	Alcobendas	25.987	03/09/2
2	Maria	Sevilla		11/04/2
3	Ester	Oviedo	30.165,98	29/11/2

- Tabla clientes

id_cliente	nombre	numero_telefono	email	direccion	ciudad	provincia
1	Monica	1234567 289	monica @email. com	Calle Felicidad	Móstoles	Madrid
2	Lorena	2893456 78	lorena@ email.co m	Calle Alegria	Barcelon a	Barcelon a
3	Carmen	2984637 59	carmen @email. com	Calle del Color	Vigo	Ponteve dra

- Tabla facturas

id_factura	numero_factura	fecha	id_zapatilla	id_empleado	id_cliente	t
1	123	11/12/2001	1	2	1	5
2	1234	23/05/2005	1	1	3	8
3	12345	18/09/2015	2	3	3	7

2. De nuevo nos hemos dado cuenta que hay algunos errores en la inserción de datos. En este ejercicios los actualizaremos para que nuestra BBDD este perfectita.

- Tabla zapatillas

En nuestra tienda no vendemos zapatillas Rosas... ¿Cómo es posible que tengamos zapatillas de

color rosa? En realidad esas zapatillas son amarillas.

- Tabla empleados

Laura se ha cambiado de ciudad y ya no vive en Alcobendas, se fue cerquita del mar, ahora vive en A Coruña.

- Tabla clientes

El Numero de telefono de Monica esta mal!!! Metimos un digito de más. En realidad su número es: 123456728

- Tabla facturas

El total de la factura de la zapatilla cuyo id es 2 es incorrecto. En realidad es: 89,91.

Happy coding

Consultas básicas 1

Queries básicas 1: SELECT, WHERE, Operaciones lógicas, IS NULL, NOT NULL

Base de datos de pruebas

Para aprender a utilizar los comandos SQL vamos a realizar ejercicios sobre una base de datos de prueba. En la siguiente imagen podemos ver la tabla `alumnas` de dicha base de datos:

id_alumna	nombre	apellido	email	telefono	direccion	ciudad	pais
1	Ana	González	ana@adalab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@adalab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@adalab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@adalab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@adalab.es	616365624	Calle Alumna 5	Paris	Francia

Como podemos observar esta tabla contiene 5 registros o entradas (una por cada alumna), y 7 columnas o atributos por cada alumna (id_alumna, nombre, apellido, email, telefono, direccion, ciudad, pais).

NOTA: En este caso, nosotras hemos definido los nombres de las columnas de la tabla tratando de no utilizar tildes o caracteres especiales, ya que en ocasiones pueden dar problemas a la hora de realizar las consultas o visualizar los datos.

Si recordáis la lección anterior, cada tabla debe tener un atributo que se usará como clave primaria para identificar únicamente cada registro de la tabla. En este caso se ha creado una columna a tal efecto llamada IDAlumna, que deberá un valor diferente para cada nueva alumna. Podemos pensar en este campo como un número de expediente que no se debe reutilizar entre alumnas. También podríamos haber incluido el atributo DNI para cada alumna y haberlo usado como clave primaria.

El resto de los atributos son más “normales” y contienen la típica información de contacto de una persona. De momento solamente nos interesa remarcar que cada atributo puede ser de un tipo diferente (como vimos también en la lección anterior): monovaluado o multivaluados, simple o compuesto. Dirección sería un ejemplo de atributo compuesto ya que puede dividirse entre el nombre de la calle y el número.

Por último, también podemos distinguir los atributos según el tipo de información que contienen: números, letras, combinación de ambos, o incluso con formatos más específicos como el email, que debe contener un @ en algún punto.

Comandos SQL.

Una vez hemos visto un ejemplo sobre la estructura que puede tener una tabla en una base de datos relacional y que tenemos algunas entradas dentro la tabla, vamos a utilizar dicho ejemplo para visualizar la utilidad de cada uno de los comandos SQL usados para manipulación de datos.

NOTA: Para probar los diferentes códigos de ejemplos de consultas SQL sobre las tablas de explicación, así como la de los ejercicios finales, recordamos que es necesario primero llamar a la base de datos sobre la que deseamos hacer las consultas para ello deberemos utilizar `USE nombre_base_datos;` o alternativamente hacer doble click sobre la columna de los diferentes esquemas que tenemos disponibles. ¡Ya que si no, no tendremos ninguna base de datos seleccionada y no podremos realizar ninguna consulta!

Consultas de Selección `SELECT` :

`SELECT` permite recuperar los datos pertenecientes a uno o varios registros de una o más tablas de la base de datos.

La sintaxis básica para realizar consultas suele ser la siguiente:

```
SELECT columna  
FROM tabla;
```

Comencemos con un ejemplo sencillo:

```
SELECT nombre  
FROM alumnas;
```

En el anterior ejemplo seleccionamos el atributo `nombre` de todas las entradas en la tabla `alumnas`. Pese a que el ejemplo parece muy sencillo, hay ciertos puntos sobre los que es importante llamar la atención:

- El primer `SELECT` indica el tipo de consulta, mientras que el siguiente campo (en este caso `nombre`) indica el atributo que se quiere seleccionar.
- Después de ellos, viene la palabra `FROM` que sirve para poder indicar la tabla de la que queremos seleccionar los datos, que en este caso será `alumnas`.
- Las palabras “clave” o especiales como `SELECT` y `FROM` no son case sensitive, es decir, da igual si las escribimos en mayúsculas o en minúsculas. Sin embargo, lo normal es escribirlas en mayúsculas para diferenciarlas bien del resto.
- *IMPORTANTE:* todas las sentencias SQL deben terminarse con un `;` para que el gestor de la base de datos pueda interpretarlo. De esta manera también podemos escribir varias sentencias seguidas, y el gestor las ejecutará una tras otra.

El resultado del anterior ejemplo será una tabla de resultados con el siguiente contenido:

nombre

Ana

Maria

Lucía

Elena

Rocío

Pasemos a un ejemplo un poco más complejo (pero no mucho) en el que vamos a seleccionar más de un atributo al mismo tiempo:

```
SELECT nombre, apellido
FROM alumnas;
```

La sintaxis del comando sigue las mismas reglas que antes (palabras especiales, el `;`, etc.) pero hemos añadido el atributo adicional que queremos seleccionar. En general, podemos seleccionar tantos atributos como queramos, solo tenemos que separarlos usando el carácter `,`.

El resultado de la consulta sería el siguiente:

nombre	apellido
Ana	González
Maria	López
Lucía	Ramos
Elena	Bueno
Rocío	García

Como es lógico, los atributos salen emparejados según el registro (fila) al que pertenecen.

Ya sabemos cómo seleccionar varios atributos dentro de una tabla, pero si queremos seleccionarlos todos, existe una manera sencilla de hacerlo sin tener que escribir toda la lista en la consulta `SELECT` :

```
SELECT *
FROM alumnas;
```

En este caso, el carácter `*` indica al gestor de la base de datos que queremos seleccionar todos los atributos al mismo tiempo. Por lo tanto, esta consulta devolverá la tabla `alumnas` entera.

Ejercicio: Realiza una consulta `SELECT` que obtenga los atributos Nombre, Teléfono y Dirección de todas las alumnas de la tabla `alumnas`.

Operador de condición `WHERE` :

Llegado este punto, podéis pensar que sacar siempre todas las filas/registros de la tabla es algo ineficiente y que no nos permite elegir o conocer los datos de alumnas concretas. En este punto es donde la instrucción `WHERE` aparece para salvar el día.

La instrucción `WHERE` sirve para seleccionar ciertos registros concretos del resultado inicial basándose en una condición que nosotras imponemos.

La sintaxis básica para realizar consultas con la instrucción `WHERE` es:

```
SELECT columna
FROM tabla
WHERE columna = condición;
```

Veámoslo con un nuevo ejemplo (a partir de ahora pondremos cada parte de la consulta SQL en una línea para mejor visualización):

```
/* Esto es un comentario en SQL */
SELECT nombre, apellido
FROM alumnas
WHERE pais = "Francia";
```

En este ejemplo hemos refinado la consulta en la que seleccionábamos los nombres y apellidos de todas las alumnas, añadiendo la condición de que sólo seleccione los de aquellas alumnas cuyo País sea Francia. Por lo tanto, el resultado que obtendremos será:

nombre	apellido
Rocío	García

Las condiciones que se pueden poner no tienen por qué estar relacionadas solamente con las columnas que estamos seleccionando al final (en este caso el Nombre y Apellido) sino que puede estar relacionada con cualquier otro atributo (en este caso el País).

Ejercicio: Realiza una consulta que obtenga los atributos Teléfono y dirección de aquellas alumnas de la tabla alumnas que se llamen Elena.

También puede quererse establecer varias condiciones al mismo tiempo. Por ejemplo, podemos seleccionar a aquellas alumnas que sean de España y que al mismo tiempo se apelliden Garcia. Para ello debemos concatenar varias condiciones en el `WHERE` usando la cláusula `AND`:

```
SELECT telefono, direccion
FROM alumnas
WHERE pais = "España"
AND apellido = "Garcia";
```

Utilizando el operador `AND` le estamos indicando al gestor de la base de datos que queremos que ambas condiciones se cumplan al mismo tiempo. De esta manera, se descartarán los registros que no cumplen alguna de ellas (aunque sí cumplan una), pero ¿qué sucedería si en el pasado ejemplo quisieramos seleccionar aquellos registros que cumplan al menos una de las dos condiciones? Para conseguirlo, podemos utilizar la cláusula `OR` en lugar de `AND`:

```
SELECT telefono, direccion
FROM alumnas
WHERE pais = "España"
OR apellido = "Garcia";
```

De esta manera, el resultado contendrá el Teléfono y la dirección de aquellas alumnas que sean de España o de aquellas cuyo Apellido sea Garcia.

Por último, podríamos querer seleccionar los registros que no cumplen cierta condición. Para ello utilizaremos el operador `NOT`, que negará la condición que venga después del mismo. Veámoslo con un ejemplo:

```
SELECT telefono, direccion  
FROM alumnas  
WHERE NOT pais = "España";
```

La anterior consulta descartará todos aquellos registros de aquellas alumnas cuyo País sea España, por lo que la tabla-resultado será:

telefono	direccion
616365624	Calle Alumna 5

Que son el Teléfono y Dirección de la única Alumna que no es de España.

Ejercicio: Realiza una consulta que obtenga los atributos Email y Nombre de aquellas alumnas de la tabla alumnas que sean de España o de Francia.

Ejercicio: Realiza una consulta que obtenga los atributos Email y Nombre de aquellas alumnas de la tabla alumnas que no se apelliden Bueno.

Observa que los atributos del `SELECT` no tienen por qué estar en el orden que están en la tabla, podemos modificar su orden como queramos (en este caso poniendo Email antes que Nombre).

Operadores de comparación:

Hasta este punto solamente hemos utilizado el operador de igualdad (`=`) para realizar comparaciones. Este operador nos sirve para comparar cadenas de texto, números, fechas, etc. Sin embargo, no siempre querremos que la condición contenida en el `WHERE` sea de igualdad. Por ejemplo, podemos querer que la condición sea de desigualdad o, cuando se trate de evaluar un número, que este sea mayor o menor que cierta cantidad. A continuación, se enumeran los operadores que se pueden utilizar a este efecto:

- Igual que (cuando queremos que el atributo sea igual que un valor): `=`
- Distinto de (cuando queremos que el atributo sea distinto a un valor): `<>`
- Menor que (cuando queremos que el atributo sea menor que un valor): `<`
- Mayor que (cuando queremos que el atributo sea mayor que un valor): `>`
- Menor o igual que (cuando queremos que el atributo sea distinto a un valor): `<=`
- Mayor o igual que (cuando queremos que el atributo sea distinto a un valor): `>=`

Vamos a ver cómo se utilizan mediante varios ejemplos.

- Seleccionar Nombre y Apellido de las alumnas cuyo Email sea `ana@adalab.es`

```
SELECT nombre, apellido  
FROM alumnas  
WHERE email = "ana@adalab.es";
```

nombre	apellido
Ana	González

- Seleccionar Ciudad y País de las alumnas cuyo teléfono sea distinto a 674459123:

```
SELECT ciudad, pais
FROM alumnas
WHERE telefono <> 674459123;
```

ciudad	pais
Madrid	España
Barcelona	España
Bilbao	España
Paris	Francia

Observa que en esta consulta el número de teléfono no se ha escrito entre comillas al no ser un atributo de tipo cadena de texto y ser un atributo de tipo número (los cuales no van entre comillas).

Ejercicio: Selecciona todos los atributos de las alumnas cuyo IDAlumna sea mayor o igual que 3.

Ejercicio: Selecciona todos los atributos de las alumnas cuyo IDAlumna sea menor o igual que 4 y que sean de Madrid.

Después de ver tantos ejemplos, os habréis dado cuenta de que comprender la función que realiza cada una de las palabras reservadas básicas (esto es `SELECT` , `WHERE` , `FROM` , `AND` , `OR`) resulta bastante intuitivo, por lo que leer consultas SQL de este nivel y visualizar los posibles resultados resulta sencillo. Poco a poco iremos complicando la lógica detrás de las consultas SQL al ir añadiendo más operadores y cláusulas, pero si logramos comprender qué realiza cada uno de ellos individualmente, no tendremos problemas en entender cómo interaccionarán dentro de una consulta real. Solo hay que ir paso a paso.

`IS NULL`

Al introducir datos en una tabla de una base de datos, normalmente se le asigna un valor a cada uno de los atributos de la tabla. Sin embargo, puede darse el caso de que para ciertas entradas no existan valores para algunos atributos. Podría ser el caso de una alumna que no tenga dirección de correo, por ejemplo. En estos casos se puede introducir el nuevo registro sin asignar un valor a esos atributos, que tomarán un valor especial denominado `NULL` (nulo).

La sintaxis para filtrar resultados con `NULL` (es decir, ausencia de valor) usando el operador `WHERE` es ligeramente diferente a lo que hacíamos para filtrar según valores específicos de los atributos. Hay que tener en cuenta que debido a que `NULL` no es igual a nada, ni siquiera a sí mismo, el uso de operadores de igualdad (`= NULL`) o desigualdad (`<> NULL`) siempre devolverá un resultado `UNKNOWN` que será rechazado por la cláusula `WHERE`.

Es por ello que si queremos seleccionar con un `WHERE` los registros para los que cierto atributo sea `NULL`, tendremos que usar el operador `IS NULL`. Imaginemos que a la hora de registrar alumnas en la tabla `alumnas`, no conociésemos la Ciudad de algunas de ellas, podríamos dejar ese atributo vacío para introducirlo más adelante. De esa manera, llegado el momento podríamos querer seleccionar aquellas alumnas sin Ciudad:

```
SELECT nombre, apellido, email  
FROM alumnas  
WHERE ciudad IS NULL;
```

La anterior consulta devolverá el Nombre, Apellido y Email de aquellas alumnas para las que no conocíamos la Ciudad a la hora de introducir sus datos en la tabla. En un futuro podríamos contactar con ellas por correo electrónico y solicitarlas que nos envíen dicha información para completar su registro.

`NOT NULL`

`IS NULL` (y algunos otros operadores que veremos más adelante en el curso) puede negarse añadiendo la palabra reservada `NOT` para invertir su comportamiento. De esa manera tendríamos `IS NOT NULL`, que buscaría aquellos registros en la base de datos que sí tengan valores almacenados para ciertos atributos.

```
SELECT nombre, apellido, email  
FROM alumnas  
WHERE ciudad IS NOT NULL;
```

En la anterior consulta buscamos aquellas alumnas que sí tienen almacenado un valor para Ciudad.

Ejercicio: Selecciona Nombre y Apellido para aquellos registros de la tabla `alumnas` que no tengan valor guardado para el atributo `Telefono`.

Enunciado Ejercicios

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisa la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` poniendo `USE tienda`; en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código (de `SELECT`, `FROM`, y `WHERE`).

La tabla `customers` tiene las siguientes columnas:

- `customer_number` : el número identificativo de las clientas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientas/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita mas espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

Ejercicio 1

Realiza una consulta `SELECT` que obtenga los nombres, teléfonos y direcciones de todas las empresas cliente de la tabla `customers`.

Ejercicio 2

Realiza una consulta que obtenga los teléfonos y direcciones de aquellas empresas de la tabla `customers` que se encuentren en USA (es su país).

Ejercicio 3

Realiza una consulta que obtenga los nombres y apellidos de las personas de contacto en cada empresa que no tenga segunda linea de dirección.

Ejercicio 4

Busca aquellos registros de la tabla `customers` que tengan un valor guardado para el campo `state`. Este atributo solo es valido para ciertos países por lo que habrá varias entradas con valor `NULL`.

Ejercicio 5

Buscar aquellos registros de la tabla `customers` que correspondan a clientes de USA pero que no tengan un valor guardado para el campo `state`.

Ejercicio 6

Selecciona el país (`country`) correspondiente a los registros de clientes con un límite de crédito (`credit_limit`) mayor que \$10000.

Resumen de la lección

- Para seleccionar la base datos a la cual queremos hacer la consulta podemos utilizar estos métodos:
 - USE nombre_base_datos
 - Hacer doble click sobre el nombre del schema de la base de datos a utilizar.
 - En el FROM escribir FROM nombre_base_datos.nombre_tabla
- Para hacer consultas utilizamos el comando SELECT () FROM :

- Para seleccionar todos los registros y atributos (filas y columnas de nuestra tabla) utilizaremos el carácter *

```
SELECT *
FROM tabla;
```

- Si deseamos extraer la información de algunas columnas en concreto en el SELECT deberemos indicar el nombre de las mismas.

```
SELECT nombre_col1, nombre_col2,....
FROM tabla;
```

- Para filtrar una consulta (o establecer condiciones de la consulta) por los valores de sus atributos (columnas) utilizaremos el WHERE:

```
SELECT *
FROM tabla
WHERE columna = condicion;
```

- Para filtrar excluyendo podemos utilizar WHERE NOT

```
SELECT * FROM tabla
WHERE NOT columna = condicion;
```

- Para filtrar por tipos de registro:

- NOT NULL: Selecciona aquellos registros cuyo valor en la columna no sea nulo.

```
SELECT * FROM tabla
WHERE columna NOT NULL;
```

- IS NULL: Selecciona únicamente aquellos registros cuyo valor sea NULL.

```
SELECT * FROM tabla
WHERE columna IS NULL;
```

- Al utilizar el WHERE podemos filtrar por más de una columna, para ello deberemos hacer uso de operadores:

- AND: Se deben cumplir las condiciones especificadas

```
SELECT * FROM tabla
WHERE columna = condicion1
AND columna2 = condicion2;
```

- OR: Se debe cumplir al menos una de las condiciones

```
SELECT * FROM tabla
WHERE columna = condicion1
OR columna2 = condicion2;
```

•

Operadores lógicos:

- <>: Para excluir aquellos valores que no cumplan la condición.

```
SELECT * FROM tabla  
WHERE columna <> condicion;
```

- =: Para seleccionar aquellos que cumplan la condición

```
SELECT * FROM tabla  
WHERE columna = condicion;
```

- > : Para seleccionar aquellas columnas que contengan valores numéricos mayores que un numero

```
SELECT * FROM tabla  
WHERE columna > condicion;
```

- >= : Para seleccionar aquellas columnas que contengan valores numéricos mayores o iguales que un numero

```
SELECT * FROM tabla  
WHERE columna >= condicion;
```

- < :Para seleccionar aquellas columnas que contengan valores numéricos menores que un numero

```
SELECT * FROM tabla  
WHERE columna < condicion;
```

- <= : Para seleccionar aquellas columnas que contengan valores numéricos menores o iguales que un numero

```
SELECT * FROM tabla  
WHERE columna <= condicion;
```

Pair programming Consultas básicas 1

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de *pair programming*. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

Como ya explicamos en la lección anterior, durante la ejecución de estas sesiones de *pair programming* vamos a trabajar sobre la base de datos `Northwind`. Esta base de datos se basa en una empresa ficticia: Northwind Traders. Contiene información acerca de las transacciones de ventas entre la empresa y sus clientes así como detalles acerca de compras de la empresa a sus proveedores. Los datos que incluye la base de datos son: tablas de inventario, pedidos, clientes, información de las empleadas, etc. Históricamente ha sido la base de muchos tutoriales y libros sobre el aprendizaje y el uso de SQL.

El día de hoy vamos a realizar ejercicios en los que practicaremos sentencias SQL sencillas usando los operadores `SELECT`, `FROM` y `WHERE`. Este tipo de sentencias nos servirán para ir extrayendo información de la base de datos e irla conociendo.

Ejercicios

1. Conociendo a las empleadas:

El objetivo de toda buena jefa (o trabajadora) de una empresa debería ser conocer bien a sus compañeras. Para ello, vamos a diseñar una consulta para obtener una lista con los datos de las empleadas y empleados de la empresa Northwind. Dicha consulta tiene que tener los campos `employee_id`, `last_name` y `first_name`.

2. Conociendo los productos más baratos:

Supongamos que en nuestro primer día en la empresa nos cuentan que la misma se encuentra en un momento de reestructuración de su negocio. Nuestras compañeras nos comentan que en estos momentos Northwind tiene demasiados productos a la venta, algunos de ellos con escaso éxito entre las clientas.

Nuestro primer encargo es ver aquellos productos (tabla `products`) cuyos precios por unidad se encuentren entre los 0 y 5 dólares, es decir, los productos más baratos.

3. Conociendo los productos de los que queremos maximizar ventas:

Por otra parte, queremos también conocer los datos de los productos que tengan exactamente un precio de 18, 19 o 20 dólares (unos valores muy concreto de precios del que la empresa quiere maximizar sus ventas en un futuro).

4. Conociendo los productos que dan más beneficios.

El rango de productos que puede dar más beneficios a la empresa podría ser el de aquellos con un precio mayor o igual a 15 dólares, pero menor o igual que 50 dólares. Selecciona los datos de ese rango de productos.

5. Conociendo los productos que no tienen precio:

Para comprobar si los datos en la tabla `products` están correctos, nos interesa seleccionar aquellos productos que no tengan precio, porque lo hayan dejado vacío al introducir los datos (`NULL`).

6. Comparando productos:

Ahora obtén los datos de aquellos productos con un precio menor a 15 dólares, pero sólo aquellos que tienen un ID menor que 10 (para tener una muestra significativa pero no tener que ver todos los productos existentes).

7. Cambiando de operadores:

Ahora vamos a hacer la misma consulta que en ejercicio anterior, pero haciendo invirtiendo el uso de los operadores y queremos saber aquellos que tengan un precio superior a 15 dólares y un ID superior a 10..

8. Conociendo los países a los que vendemos:

A Northwind le interesa conocer los datos de los países que hacen pedidos (`orders`) para focalizar el negocio en esas regiones y al mismo tiempo crear campañas de marketing para conseguir mejorar en las otras regiones. Realiza una consulta para obtener ese dato.

Happy coding

Consultas básicas 2

Queries básicas 2: ORDER BY, DISTINCT, LIMIT, OFFSET, BETWEEN, IN, AS

Después de comprender las consultas SQL más básicas (aquellas compuestas `SELECT`, `FROM` y `WHERE` únicamente), en esta lección vamos a aprender a utilizar algunos comandos e instrucciones que se pueden añadir a las consultas SQL para dotarlas de funcionalidades más avanzadas y complejas.

ORDER BY

`ORDER BY` es un operador adicional que se incluye después del `WHERE` y sirve para indicar el orden que se desea que tenga la tabla Resultado final. Es decir, con `ORDER BY` le indicamos al intérprete SQL la o las columnas por las que queremos que se ordene el resultado que veremos por pantalla. Volviendo al ejemplo sobre la tabla `alumnas` que definimos en las secciones anteriores:

id_alumna	nombre	apellido	email	telefono	direccion	ciudad	pais
1	Ana	González	ana@adalab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@adalab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@adalab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@adalab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@adalab.es	616365624	Calle Alumna 5	Paris	Francia

```
SELECT *
FROM alumnas
ORDER BY apellido DESC;
```

En la consulta anterior los resultados seleccionados se ordenarán siguiendo los valores para la columna Apellido. La palabra reservada DESC se puede añadir para invertir el comportamiento de ORDER BY para que la ordenación sea decreciente (de mayor a menor) en vez de creciente (de menor a mayor). De esta forma, estamos ordenando los registros según el Apellido en orden alfabético decreciente (al ser una cadena de texto). El resultado será el siguiente:

id_alumn a	nombre	apellido	email	telefono	direccion	ciudad	pais
3	Lucía	Ramos	lucia@ad lab.es	6744591 23	Calle Alumna 3	Valencia	España
2	Maria	López	maria@ad lab.es	6896563 22	Calle Alumna 2	Barcelon a	España
1	Ana	González	ana@ad lab.es	6547852 14	Calle Alumna 1	Madrid	España
5	Rocío	García	rocio@ad lab.es	6163656 24	Calle Alumna 5	Paris	Francia
4	Elena	Bueno	elena@ad lab.es	6285465 77	Calle Alumna 4	Bilbao	España

Normalmente la columna que se usa como criterio de ordenación también se incluye entre las seleccionadas en el SELECT , pero no es obligatorio que sea así. Podríamos ordenar las alumnas por su Apellido y no seleccionar el mismo para el resultado final.

Ejercicio: En la tabla alumnas selecciona los valores que toma el atributo Ciudad para aquellas alumnas que son España y ordénalos por orden alfabético (según la Ciudad).

DISTINCT

El operador DISTINCT nos ayudará a listar en la tabla-resultado únicamente los valores diferentes que toma un atributo en la tabla. De esta manera se eliminarán del resultado los valores duplicados.

```
SELECT DISTINCT pais
FROM alumnas;
```

En el caso anterior la consulta seleccionaría en primer lugar el atributo Pais de todos los registros/filas de la tabla Alumna. Después, gracias al operador DISTINCT se seleccionarán los valores diferentes que toma el atributo Pais. El resultado será:

país

España

Francia

Se puede ver como el operador `DISTINCT` ha servido para mostrar cada posible valor que toma País una sola vez. Pese a haber muchas alumnas de España, este valor se muestra una única vez.

`DISTINCT` puede usarse también cuando elegimos más de un atributo con `SELECT` :

```
SELECT DISTINCT nombre, apellido  
FROM alumnas;
```

En el ejemplo superior la consulta elige los atributos Nombre y Apellido para todas las alumnas de la tabla alumnas. Posteriormente el operador `DISTINCT` afectará a todos los atributos del `SELECT`, por lo que la tabla resultado final reflejará todas las combinaciones diferentes de Nombre y Apellido (en este caso no se repiten en ningún caso, por lo que sacará a todas las alumnas). Si hubiese en la tabla alumnas una pareja de alumnas con el mismo nombre y apellido, esa combinación solo se mostraría una vez en la tabla resultado. El resultado será:

nombre	apellido
Ana	González
Maria	López
Lucía	Ramos
Elena	Bueno
Rocío	García

Ejercicio: En la tabla alumnas selecciona los distintos valores que toma el atributo Ciudad para aquellas alumnas que son de España.

LIMIT

La cláusula `LIMIT` se utiliza para especificar el número de registros que queremos que contenga la tabla resultado, ya que no siempre querremos ver todos los resultados posibles. En este caso, vamos a visualizar una consulta ejemplo realizada sobre una tabla llamada Empleadas:

idempleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Ahora vamos a emplear `ORDER BY` combinado con `LIMIT` para conseguir el objetivo de ordenar a las empleadas en orden decreciente de apellido y quedarnos con las 10 primeras empleadas en la lista:

```
SELECT nombre, apellido, salario
FROM empleadas
ORDER BY apellido DESC
LIMIT 10;
```

De esta manera, en tablas con un número muy alto de entradas, podemos seleccionar no visualizar todos los resultados y quedarnos con un conjunto de datos más "manejable". El resultado de la consulta anterior sería:

nombre	apellido	salario
Lucía	Ramos	3000
Maria	López	4000
Ana	González	2500
Rocío	García	1500
Elena	Bueno	5000

En este caso únicamente nos devuelve 5 valores, ya que la tabla de empleadas solo contiene 5 registros.

Otro ejemplo de `LIMIT` podría darse en una empresa que quiera ordenar a sus empleadas en orden decreciente de salario para así luego poder seleccionar los datos de las 10 empleadas que más cobran. Para realizar esta consulta, podría valerse del operador `ORDER BY` (como hemos visto hace un momento) combinado a su vez con `LIMIT`:

idempleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

```
SELECT nombre, apellido, salario
FROM empleadas
ORDER BY salario DESC
LIMIT 10;
```

En esta consulta `ORDER BY` indica que los resultados se ordenarán siguiendo los valores de la columna Salario, `DESC` se añade para indicar que la ordenación se haga siguiendo un orden decreciente (de mayor a menor salario). Por su parte `LIMIT 10` implica que solo se elegirán los 10 primeros resultados devueltos por el resto de la consulta.

El resultado de la consulta anterior sería el siguiente:

nombre	apellido	salario
Elena	Bueno	5000
Maria	López	4000
Lucía	Ramos	3000
Ana	González	2500
Rocío	García	1500

Ejercicio: En la tabla Empleadas anterior selecciona todos los datos de las 3 empleadas que menos cobren y cuyo país sea España.

Ejercicio: En la tabla Empleadas selecciona los 2 primeros valores que toma el atributo país para aquellas empleadas que son de España cuando se los ordena por orden alfabético (según el nombre de las empleadas).

OFFSET

El comando `OFFSET` se puede utilizar para descartar los primeros resultados de una consulta y solo ver aquellos a partir de la posición indicada por `OFFSET`.

En los ejemplos previos en los que hemos usado `LIMIT`, podríamos querer conocer las empleadas que más cobran descartando las 3 primeras (por la razón que sea). Se realizaría de la manera siguiente:

```
SELECT nombre, apellido, salario
FROM empleadas
ORDER BY salario DESC
LIMIT 10
OFFSET 3;
```

El resultado de la consulta anterior sería:

nombre	apellido	salario
Ana	González	2500
Rocío	García	1500

Ejercicio: En la tabla Empleadas selecciona el Nombre y Apellido de la 2^a empleada que más cobre en España.

BETWEEN

El operador `BETWEEN` es equivalente a una combinación de los operadores lógicos de comparación `<=` y `>=`. Es decir, como bien indica la traducción del comando al español (*entre*), la cláusula seleccionará las entradas de la tabla cuyos valores se encuentren dentro en un rango de valores. Veámoslo con un ejemplo:

```
SELECT nombre, apellido
FROM alumnas
WHERE id_alumna BETWEEN 3 AND 5;
```

El ejemplo anterior seleccionará el Nombre y Apellido de aquellas alumnas cuyo IDAlumna se encuentre entre los valores 3 y 5 (ambos incluidos, al ser equivalente a un `>=` y un `<=`). La tabla Resultado será:

nombre	apellido
Lucía	Ramos
Elena	Bueno
Rocío	García

Ejercicio: Utilizando el ejemplo de la tabla Empleadas, selecciona los atributos Nombre y Apellido para aquellas empleadas de España que tengan un Salario entre 1000 y 3000 €.

IN

Otra condición bastante útil y común puede consistir en seleccionar los registros para los que el valor de un determinado atributo se encuentre entre los indicados en una lista. Si esa es nuestra situación, podemos utilizar el operador `IN`:

```
SELECT nombre, apellido  
FROM alumnas  
WHERE Ciudad IN ('Madrid', 'Valencia', 'Barcelona');
```

El ejemplo previo ha seleccionado el Nombre y Apellido de aquellas alumnas cuya Ciudad sea cualquiera de las tres especificadas en la lista. Otro ejemplo donde este tipo de condición podría ser útil sería el de en una empresa que quisiese conocer todas las empleadas que han participado en una lista de proyectos.

El resultado de la consulta anterior sería:

```
SELECT apellido, salario  
FROM empleadas  
WHERE pais IN ('España', 'Alemania');
```

nombre	apellido
Ana	González
Maria	López
Lucía	Ramos

Ejercicio: Utilizando el ejemplo de la tabla Empleadas, selecciona Apellido y Salario de aquellas empleadas que viven en España o Alemania.

AS

En SQL pueden emplearse alias para cambiar el nombre de las columnas de la tabla Resultado o de las propias tablas Resultado. Es un cambio temporal que no afecta a los nombres originales de la tabla o de los atributos.

```
SELECT nombre AS Name, apellido AS Surname  
FROM alumnas AS Students;
```

En el ejemplo anterior hemos creado unos alias para las columnas resultado. En este caso hemos decidido cambiar los nombres con lo que las visualizábamos originalmente por sus traducciones en inglés, pero cualquier otro cambio es posible. Nosotras hemos cambiado el nombre de los dos atributos seleccionado y de la tabla, pero no es necesario hacerlo para todos a la vez, podría cambiarse solo el nombre de la tabla o de una sola de las columnas.

El resultado de la consulta anterior sería:

Name	Surname
Ana	González
Maria	López
Lucía	Ramos
Elena	Bueno
Rocío	García

Ejercicio: Utilizando el ejemplo de la tabla Empleadas, selecciona Apellido y Salario de aquellas empleadas que viven en España y renombra las columnas resultado a Alumna y Sueldo respectivamente.

Resumen Consultas básicas 2

- ORDER BY: Permite ordenar las consultas, por defecto las ordena de forma ascendente.

- Formas de ordenar:

- Ascendente: ASC . Ordena del valor más pequeño a más grande.

```
SELECT nombre_col1, nombre_col2,....  
FROM tabla  
ORDER BY ASC;
```

- Descendente: DESC. ordena del valor más grande al más pequeño. Se tiene que indicar expresamente.

```
SELECT nombre_col1, nombre_col2,....  
FROM tabla  
ORDER BY DESC;
```

- DISTINCT: Permite filtrar la columna por aquellos valores que sean únicos, es decir, nos quita los duplicados.

- Se coloca despues del SELECT.
- Podemos utilizarlo para más de un atributo (columna).

```
SELECT DISTINCT nombre_col1, nombre_col2,....  
FROM tabla  
```
```

- LIMIT numero\_valores : Limita el número de valores seleccionados de la consulta. Por defecto MySQL selecciona un máximo de 1000 registros.

- Muy util su uso cuando estamos explorando bases de datos con muchos registros.
- Siempre se coloca al final de la consulta.

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla

LIMIT numero_valores
```
```

- OFFSET numero_valores: Se utiliza para descartar los primeros (numero_valores) indicados en la consulta.

- No permite utilizar números negativos.
- Obligatoriamente debemos utilizarlo junto a LIMIT.

```
SELECT DISTINCT nombre_col1, nombre_col2,....  
FROM tabla  
LIMIT numero_valores1  
OFFSET numero_valores2  
```
```

- BETWEEN: Es una condición del WHERE. Permite filtrar por rangos de datos, considerando la columna por la que estamos filtrando.

- Solo acepta valores numéricos.
-

Es equivalente a filtrar la columna numérica utilizando  $\leq v \geq$

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla
WHERE columna BETWEEN valor1 AND valor2
```
```

- AS: Se utiliza para darle un alias (nombre) a una columna de forma temporal, no modifica el nombre real de la columna.

```
SELECT DISTINCT nombre_col1 AS nombre_columna1_alias, nombre_col2 AS nombre_columna2_a  
FROM tabla  
WHERE columna;  
```
```

- IN: Nos permite filtrar utilizando uno o varios elementos de la columna por la que estamos filtrando.

- Se utiliza en conjunción con el WHERE.
- Se puede replicar este comportamiento utilizando OR anidados.

```
SELECT DISTINCT nombre_col1, nombre_col2,....
FROM tabla
WHERE columna IN(valor1, valor2);
```
```

Pair programming Consultas básicas 2

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de pair programming. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind.

Hoy vamos a realizar ejercicios en los que practicaremos sentencias SQL algo más avanzadas en las que introduciremos el uso de operadores como `ORDER BY` , `DISTINCT` , `LIMIT` , `OFFSET` , `BETWEEN` , `IN` y `AS` . Gracias al uso de estos operadores seréis capaces de seleccionar información mucho más específica de la base de datos, la cual nos va a servir para realizar un análisis más en profundidad del negocio de la empresa.

Ejercicios

1. Conociendo el tipo de productos que vendemos en Northwind:
Crea una consulta que muestre los primeros 10 productos según su ID y que nos indique el nombre de dichos productos y sus precios.
2. Ordenando los resultados:
Ahora realiza la misma consulta pero que nos muestre los últimos 10 productos según su ID de manera descendiente.
3. Que pedidos tenemos en nuestra BBDD:
Últimamente ha habido algo de descontrol en la empresa a la hora de controlar los pedidos. Nos interesa conocer qué pedidos distintos hemos tenido (eliminando entradas duplicadas según su ID en la tabla `order_details`).
4. Los dos primeros pedidos:
Como el número de pedidos es demasiado alto para visualizarlo de manera práctica en la consulta anterior, vamos a limitar los resultados sólo a los 2 primeros pedidos para verlos más en detalle. Sin embargo, en la tabla `OrderDetails` pueden existir varios registros para cada pedido, por lo que no nos servirá con un uso normal del comando `LIMIT`.
5. Qué pedidos han gastado más:
Una vez hemos inspeccionado el tipo de pedidos que tenemos en la empresa, desde la dirección nos piden conocer los 3 pedidos que han supuesto un mayor coste económico total para la empresa.
Calcúlalo y dale el alias `ImporteTotal`. Nota: Utiliza `unit_price` y `quantity` para calcular el importe total.
6. Los pedidos que están entre las posiciones 5 y 10 de nuestro *ranking*:
Ahora, no sabemos bien por qué razón, desde el departamento de Ventas nos piden seleccionar el ID de los pedidos situados entre la 5 y la 10 mejor posición en cuanto al coste económico total `ImporteTotal`.
7. Qué categorías tenemos en nuestra BBDD:
De cara a ver cómo de diversificado está el negocio, se nos solicita una lista de las categorías que componen los tipos de pedido de la empresa. Queremos que la lista de resultado sea renombrada como "NombreDeCategoria".
8. Selecciona envíos con retraso:
Nos hacen llegar desde la dirección la preocupación acerca del cumplimiento de las fechas de envío. Últimamente se están dando retrasos en muchas entregas y por ello se busca realizar la acción preventiva de enviar los paquetes con varios días adicionales de antelación. Para comenzar a planear esos envíos anticipados, nos piden conocer cuál sería la fecha de envío (`ShippedDate`) de los pedidos almacenados en la base de datos, si estos sufrieran un retraso de 5 días. Nos piden mostrar la nueva fecha renombrada como `FechaRetrasada`.
Pista Para realizar lo anterior, busca documentación de la función `DATE_ADD` para MySQL.
9. Selecciona los productos más rentables:
Gracias a un análisis realizado en los últimos meses en la empresa, se ha comprobado que el rango de productos que puede dar más beneficios parece ser el de aquellos con un precio mayor o igual a 15 dólares, pero menor o igual que 50 dólares. Selecciona los datos de ese rango de productos usando el operador `BETWEEN`.
10. Selecciona los productos con unos precios dados:
Queremos conocer los datos de los productos que tengan exactamente un precio de 18, 19 o 20 dólares (un rango muy concreto de precios del que la empresa quiere maximizar sus ventas en un

futuro). Usa `IN` para conseguirlo de manera eficiente.

Happy coding

Consultas avanzadas 1

Queries avanzadas 1: Funciones agregadas

Funciones agregadas y SELECT

Las funciones agregadas permiten hacer operaciones con los datos de la base de datos que obtenemos con la consulta `SELECT` original. Son funciones que toman columnas de la tabla resultado y calculan u obtienen datos acerca de ellas. Ejemplos del tipo de cosas que podemos calcular son el máximo o mínimo de una columna, su valor medio, el número de filas que contiene el resultado, etc.

Es importante conocer que estas funciones agregadas solo pueden utilizarse en el campo `SELECT` de la consulta (o en el campo `HAVING` que veremos más adelante en el curso).

MIN, MAX

Tal y como la intuición nos indica, la función `MIN()` devuelve como resultado únicamente el valor más pequeño de una columna mientras que la función `MAX()` por el contrario devuelve como resultado el valor mayor de la columna escogida.

Veamos su funcionamiento sobre un ejemplo sobre la tabla `alumnas`.

id_alumna	nombre	apellido	email	telefono	direccion	ciudad	pais
1	Ana	González	ana@adalab.es	654785214	Calle Alumna 1	Madrid	España
2	Maria	López	maria@adalab.es	689656322	Calle Alumna 2	Barcelona	España
3	Lucía	Ramos	lucia@adalab.es	674459123	Calle Alumna 3	Valencia	España
4	Elena	Bueno	elena@adalab.es	628546577	Calle Alumna 4	Bilbao	España
5	Rocío	García	rocio@adalab.es	616365624	Calle Alumna 5	Paris	Francia

```
SELECT MIN(id_alumna) AS IDMenor  
FROM alumnas;
```

En este caso la consulta está seleccionando el ID de todas las entradas de la tabla alumnas. Posteriormente, la función `MIN()` busca en dicha columna y se queda con el menor valor que encuentre. Dado que el resultado que muestra la columna ya no serían todos los valores del ID sino el menor valor existente en la tabla, hemos decidido usar el operador `AS` para crear un alias para el resultado que sea más descriptivo (esto es opcional). El resultado obtenido sería el siguiente:

IDMenor
1

Ejercicio: Selecciona el número de ID más alto de la tabla alumnas y asígnale un alias que sea explicativo.

Estas funciones también se pueden usar con atributos y datos alfanuméricos. En ese caso el operador `MIN` se quedará con el resultado que fuese el primero si ordenásemos los registros por orden alfabético (de la A a la Z). Si usásemos `MAX` sería similar pero quedándose con el último resultado.

Ejercicio: Selecciona el último nombre de alumna si ordenásemos la columna en orden alfabético.

SUM, AVG, y COUNT

Algunas funciones agregadas más complejas que `MIN()` y `MAX()` son `SUM()`, `AVG()` y `COUNT()`, cuyos nombres son de nuevo bastante descriptivos.

El operador `SUM()` realiza la suma de todas las entradas en la columna indicada. Si vamos al ejemplo de la tabla empleadas que contiene una columna Salario para cada una de las empleadas, `SUM()` resultaría útil para conocer datos como la cantidad de dinero que una empresa está gastando en los salarios de sus empleadas:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

```
SELECT SUM(salario) AS TotalSalarios  
FROM empleadas;
```

El resultado obtenido de la consulta anterior será el siguiente:

```
TotalSalarios
```

```
16000
```

De esta forma, el resultado será un solo valor denominado TotalSalarios, que contendrá la suma de los salarios de todas las empleadas de la tabla empleadas.

El operador `AVG()` por su parte sirve para calcular el valor medio (average) del atributo especificado. Volviendo al ejemplo anterior, nos serviría para conocer el salario medio de las empleadas de la empresa:

```
SELECT AVG(salario) AS SalarioMedio  
FROM empleadas;
```

El resultado de la consulta anterior será el siguiente:

```
SalarioMedio
```

```
3200
```

Por último, `COUNT()` es una función que devuelve el número de registros (filas) tiene la tabla Resultado original:

```
SELECT COUNT(salario) AS SalariosAltos  
FROM empleadas  
WHERE salario >= 3000;
```

El resultado de la consulta anterior será:

```
SalariosAltos
```

```
3
```

En el ejemplo anterior, la función `COUNT()` ha resultado útil para conocer cuántas empleadas de la empresa tienen un salario mayor a 3000€ (gracias a la condición establecida con `WHERE`).

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisa la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla customers tiene las siguientes columnas:

- `customer_number` : el número identificativo de las clientas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientas/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita mas espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

EJERCICIO 1

Realiza una consulta `SELECT` que obtenga el número identificativo de cliente más bajo de la base de datos.

EJERCICIO 2

Selecciona el límite de crédito medio para los clientes de España.

EJERCICIO 3

Selecciona el numero de clientes en Francia.

EJERCICIO 4

Selecciona el máximo de crédito que tiene cualquiera de los clientes del empleado con número 1323.

EJERCICIO 5

¿Cuál es el número máximo de empleado de la tabla customers?

Pair programming Consultas avanzadas 1

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de pair programming. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind.

El día de hoy vamos a realizar ejercicios en los que practicaremos sentencias SQL usando los operadores `MIN` , `MAX` , `SUM` , `AVG` , `COUNT` para agregar la información extraída de las bases de datos. De esta manera podremos obtener información algo más general acerca de los registros y atributos de las tablas.

Ejercicios

1. Productos más baratos y caros de nuestra BBDD:

Desde la división de productos nos piden conocer el precio de los productos que tienen el precio más alto y más bajo. Dales el alias `lowestPrice` y `highestPrice` .

2. Conociendo el numero de productos y su precio medio:

Adicionalmente nos piden que diseñemos otra consulta para conocer el número de productos y el precio medio de todos ellos (en general, no por cada producto).

3. Sacad la máxima y mínima carga de los pedidos de UK:

Nuestro siguiente encargo consiste en preparar una consulta que devuelva la máxima y mínima cantidad de carga para un pedido (freight) enviado a Reino Unido (United Kingdom).

4. Qué productos se venden por encima del precio medio:

Después de analizar los resultados de alguna de nuestras consultas anteriores, desde el departamento de Ventas quieren conocer qué productos en concreto se venden por encima del precio medio para todos los productos de la empresa, ya que sospechan que dicho número es demasiado elevado.

También quieren que ordenemos los resultados por su precio de mayor a menor.

NOTA: para este ejercicio puedes necesitar dos consultas separadas

5. Qué productos se han descontinuado:

De cara a estudiar el histórico de la empresa nos piden una consulta para conocer el número de productos que se han descontinuado. El atributo `Discontinued` es un booleano: si es igual a 1 el producto ha sido descontinuado.

6. Detalles de los productos de la query anterior:

Adicionalmente nos piden detalles de aquellos productos no descontinuados, sobre todo el `ProductID` y `ProductName`. Como puede que salgan demasiados resultados, nos piden que los limitemos a los 10 con ID más elevado, que serán los más recientes. No nos pueden decir del departamento si habrá pocos o muchos resultados, pero lo limitamos por si acaso.

Happy coding

Consultas avanzadas 2

Queries avanzadas 2: GROUP BY, HAVING y CASE

Las funciones agregadas (`MAX` , `MIN` , `COUNT` , `AVG` , etc.) que hemos aprendido a utilizar en la lección anterior tienen mucho potencial por sí mismo. Hemos visto como podemos calcular el salario mínimo de las empleadas de una empresa, o incluso el salario medio de las empleadas de cierto país. Sin embargo, su uso resultaría mucho más interesante si tuviésemos la capacidad de agrupar o seleccionar previamente los resultados según los valores de algunos de los atributos de la tabla, antes de calcular las funciones agregadas. En esta lección veremos algunos operadores como `GROUP BY` que nos permitirán añadir esta funcionalidad a nuestras consultas SQL.

GROUP BY

La instrucción `GROUP BY` se utiliza para agrupar las filas resultado según los valores de uno de sus atributos. La salida de la instrucción será un resumen del resto de atributos, debido a ello `GROUP BY` se utiliza a menudo con funciones agregadas como `COUNT()` , `MAX()` , etc., que se aplican a esas otras columnas de manera independiente. Volviendo a nuestro ejemplo recurrente sobre los salarios de las empleadas de una empresa, el uso de `GROUP BY` en conjunto con las funciones agregadas podría hacer a estas últimas aún más interesantes, ya que podríamos agrupar a las empleadas por país, para posteriormente calcular el salario mínimo de las empleadas para *cada* país de los existentes en la tabla, obteniendo la información de forma mucho más rápida y práctica que en el caso en los que no usábamos `GROUP BY` , cuando teníamos que ir seleccionando los datos país por país mediante el uso de `WHERE` .

Todo esto puede parecer algo abstracto, pero lo entendemos mejor viendo su funcionamiento paso a paso. De nuevo consideremos la tabla empleadas perteneciente a una compañía:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia

Si quisieramos seleccionar aquellas empleadas de cada país que tengan un salario alto (mayor a 3000€ mensuales), podríamos usar GROUP BY de la siguiente manera:

```
SELECT COUNT(salario) AS SalariosAltosPorPais, pais
FROM empleadas
WHERE salario >= 3000
GROUP BY pais;
```

El resultado de dicha consulta sería el siguiente:

SalariosAltosPorPais	pais
3	España

En la tabla del empleadas existían 3 empleadas diferentes en España cuyo sueldo era mayor de 3000 €, mientras que en Francia no había ninguna empleada con un salario por encima de ese valor.

Como la consulta anterior puede resultar algo compleja de interpretar de primeras vamos a ver en diferentes pasos como se ha realizado esta consulta.

1. De la tabla empleadas se seleccionan las columnas salario y pais.

```
SELECT salario , pais
FROM empleadas;
```

Que devolvería lo siguiente:

salario	pais
2500	España
4000	España
3000	España
5000	España
1500	Francia

1. Aplicamos ahora el filtro para aquellos salarios por encima de los 3000 euros:

```
SELECT salario, pais
FROM empleadas
WHERE salario >= 3000 ;
```

Que devolvería lo siguiente:

salario	pais
4000	España
3000	España
5000	España

Fijate que ahora tenemos menos resultados que antes.

- Como queremos contar el numero de salarios por encima de 3000 euros, aplicamos ahora el COUNT, donde ademas le ponemos un alias al conteo, y el GROUP BY, para que nos agrupe por pais y nos realice el conteo de casos:

```
SELECT COUNT(salario) AS SalariosAltosPorPais, pais
FROM empleadas
WHERE salario >= 3000
GROUP BY pais;
```

Que nos devuelve de nuevo lo que teníamos originalmente.

SalariosAltosPorPais	pais
3	España

En forma resumida el comportamiento de la consulta anterior es el siguiente: Como solo queremos seleccionar los registros de las empleadas con salarios altos, se introduce una condición WHERE a tal efecto. Despues, para agrupar las empleadas con salarios altos segun su pais se introduce la cláusula GROUP BY que agrupa los resultados segun esa columna. Adicionalmente, se utiliza la función agregada COUNT() sobre la columna salario, que se aplicara para cada uno de los grupos de manera independiente, devolviendo el número de registros/filas para cada grupo con el mismo país:

NOTA: No hay que olvidar que cuando se usa GROUP BY en una consulta, en el SELECT sólo podrán seleccionarse las columnas que se especifican en la cláusula GROUP BY o en su defecto, otras columnas sobre las que aplicamos funciones agregadas (como COUNT, AVG, MAX, etc.).

Ejemplo: Trabajando de nuevo sobre la tabla empleadas que hemos definido anteriormente, calcula el salario medio por ciudad de las empleadas españolas:

```
SELECT ciudad, AVG(salario) AS SalarioMedio
FROM empleadas
WHERE pais = "España"
GROUP BY ciudad;
```

El resultado de la consulta anterior sería el siguiente:

ciudad	SalariosMedio
Madrid	2500
Barcelona	4000
Valencia	3000
Bilbao	5000

Ejercicio: ¿Cuantas empleadas hay en cada ciudad?

HAVING

La cláusula `HAVING` se agregó en su momento al lenguaje SQL debido a que `WHERE` no se puede usar con funciones agregadas o grupos (ya que se aplica con anterioridad a la creación de dichos grupos). Por lo tanto, `HAVING` se usa para imponer condiciones a los grupos creados con `GROUP BY` una vez estos ya se han creado.

Por ejemplo, en el caso de la tabla empleadas, podríamos preguntarnos por el salario medio de las empleadas para cada país, pero solo para los países que tengan al menos 3 empleadas:

```
SELECT AVG(salario) AS SalariosMediosPais, pais
FROM empleadas
GROUP BY pais
HAVING COUNT(*) >= 3;
```

El resultado de la consulta anterior sería el siguiente:

SalariosMediosPais	pais
3625	España

La anterior consulta agrupa las empleadas por país para después calcular la función agregada `AVG` sobre sus salarios. De dicha manera, devuelve el salario medio por país. Adicionalmente a esto, gracias a la cláusula `HAVING`, estamos imponiendo la condición de que solo se muestren los resultados para los grupos (países en este caso) con más de 3 registros (`empleadas`). Se ha utilizado el `*` para indicar que se aplique a todas las columnas agrupadas bajo `pais`, dado que en este caso solo existe `salario`, hubiese servido también `HAVING COUNT(salario) >= 3`. El resultado obtenido para la tabla ejemplo debería ser de 3625 euros.

Ejercicio: Trabajando de nuevo sobre la tabla `empleadas` que hemos definido anteriormente, selecciona los nombres de las ciudades con una o más empleadas.

CASE

CASE sirve para crear consultas SQL en las que se realicen diferentes acciones en función de si los datos cumplen unas condiciones u otras. De esta manera podemos tener en cuenta diferentes escenarios/casos y tomar una acción diferente dependiendo de en cual nos encontremos. Por ejemplo:

```
SELECT
CASE
    WHEN salario < 2000 THEN "Bajo"
    ELSE "Alto"
END AS RangoSalario
FROM empleadas;
```

En este ejemplo se realizan diferentes acciones en función del salario de las empleadas. Usando CASE podemos comprobar si el salario de las empleadas está por encima o por debajo de un umbral que nosotros hemos definido (2000 euros). En función de si se cumple esa condición, podemos mostrar por pantalla un texto descriptivo u otro, en este caso 'Bajo' si está por debajo del umbral o 'Alto' si está por encima del umbral. Finalmente, le damos el alias RangoSalario a la columna resultado del SELECT CASE para saber qué significa dicha etiqueta (este alias es opcional y no es necesario ponerlo).

El resultado sería el siguiente:

RangoSalario

Alto

Alto

Alto

Alto

Bajo

Es importante añadir el operador END al final de la definición de la casuística, para indicarle al intérprete de SQL que ya ha terminado la sentencia CASE . Adicionalmente se le asigna un alias o nombre a dicha casuística (en este caso la hemos llamado RangoSalario), que será el nombre que tenga la columna resultado.

CASE puede usarse también en aquellos casos más complejos en los que quisiéramos tener en cuenta varias condiciones o umbrales. Para conseguirlo podemos encadenar múltiples WHEN para tener en cuenta esas múltiples opciones:

```

SELECT
CASE
    WHEN salario < 2000 THEN "Bajo"
    WHEN salario > 3000 THEN "Alto"
    ELSE "Medio"
END AS RangoSalario, salario
FROM empleadas;

```

En esta nueva consulta se han determinado 2 umbrales diferentes para el salario, 2000 y 3000 euros, para dividir a las empleadas en tres categorías: 'Bajo', 'Medio' y 'Alto'. De nuevo se ha finalizado el `CASE` añadiendo el operador `END` y un alias para el mismo (RangoSalario). También hemos añadido a los resultados seleccionados la columna salario.

El resultado de la consulta anterior sería el siguiente:

RangoSalario	salario
Medio	2500
Alto	4000
Medio	3000
Alto	5000
Bajo	1500

Adicionalmente, se pueden introducir condiciones con `CASE` dentro de otras sentencias `CASE`, creando subcasos dentro de casos:

```

SELECT
CASE
    WHEN salario < 2000 THEN "Bajo"
    ELSE
        CASE WHEN salario > 3000 THEN "Alto"
        ELSE "Medio"
        END
    END RangoSalario
FROM empleadas;

```

Esta última consulta tiene el mismo objetivo que la anterior (dividir los salarios de las empleadas en tres rangos/categorías diferentes), aunque en este caso hace uso de `CASE` anidados. El primer `CASE` comprueba una condición para el salario y si no se cumple (indicamos ese resto de casos con `ELSE`) abrimos otra sentencia `CASE` en la que comprobamos otra condición, que tiene su `ELSE` también por si no se cumple.

El resultado de la consulta anterior sería el siguiente:

RangoSalario	salario
Medio	2500
Alto	4000
Medio	3000
Alto	5000
Bajo	1500

El resultado como vemos es el mismo que en caso en el que habíamos usado un solo `CASE` con dos umbrales (`WHEN`). Sin embargo es interesante que os deis cuenta de que el punto diferenciador a la hora de usar varios `CASE` separados con el operador `ELSE` es que si el primer `CASE` se cumple, el resto de condiciones dentro del `ELSE` no se comprobarán. En el primer ejemplo, ambos umbrales estaban dentro del mismo `CASE` (sin separar por `ELSE`) por lo que siempre se comprobaban todas las condiciones.

NOTA: Cada `CASE` debe tener su `END` asociado (y de manera opcional su alias). Es importante también añadir los operadores `ELSE` para que el interprete sepa como continuar o actuar cuando no se cumpla ninguna de las condiciones consideradas en los `CASE`.

`CASE` puede aparecer en el `SELECT`, pero también dentro del `WHERE`, `HAVING`, `GROUP BY` y otras funciones escalares. De esta manera, los valores que tomarán de argumento estas cláusulas pueden ser diferentes según se den ciertas condiciones que se comprueban con `CASE`.

Un ejemplo del uso de `CASE` dentro del `WHERE` sería la siguiente consulta SQL:

```
SELECT nombre, apellido
FROM empleadas
WHERE salario >
(SELECT CASE
    WHEN pais = "España" THEN 1000
    WHEN pais = "Francia" THEN 2000
    ELSE 1500
    END);
```

En ella estamos seleccionando el nombre y apellidos únicamente de aquellas empleadas cuyo salario sea mayor que 1000 euros en el caso de que sean de España, mayor que 2000 cuando sean de Francia y de 1500 euros en el resto de casos. Es importante ver que aunque el `CASE` esté en el `WHERE`, el `ELSE`, y el `END` también se incluyen.

El resultado de la consulta anterior sería el siguiente:

nombre	apellido
Ana	González
Maria	López
Lucía	Ramos
Elena	Bueno

Ejercicios: Trabajando de nuevo sobre la tabla empleadas que hemos definido anteriormente:

- Selecciona nombre y apellidos únicamente de aquellas empleadas que sean de España y asígnale una etiqueta que sea 'Española'.
- Selecciona aquellas empleadas que sean de España y asígnale una etiqueta que sea 'Española'. Al resto de empleadas asígnale la etiqueta Extranjera.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisita la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla `customers` tiene las siguientes columnas:

- `customer_number` : el número identificativo de las cuentas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientes/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita más espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

EJERCICIO 1

Realiza una consulta `SELECT` que seleccione el número de cada empleado de ventas, así como el número de clientes distintos que tiene cada uno.

EJERCICIO 2

Selecciona el número de cada empleado de ventas que tenga más de 7 clientes distintos.

EJERCICIO 3

Selecciona el número de cada empleado de ventas, así como el número de clientes distintos que tiene cada uno. Asigna una etiqueta de "AltoRendimiento" a aquellos empleados con más de 7 clientes distintos.

EJERCICIO 4

Selecciona el número de clientes en cada país.

EJERCICIO 5

Selecciona aquellos países que tienen clientes de más de 3 ciudades diferentes.

Repaso consultas avanzadas 2

- GROUP BY: Agrupa los datos en función de los valores del atributo (o atributos) que introduzcamos en la clausula GROUP BY.
 - Podemos aplicar el group by para más de una columna(atributo).
 - Cuando los group by da errores posiblemente se debe a que has incluido en el select una columna que tiene más registros que el numero total de registros devueltos por la instrucción group by.
 - Podemos utilizarlo sin funciones agregadas y con funciones agregadas.
 - Suele ser de las últimas instrucciones que escribimos en las consultas.
 - Podemos utilizar el WHERE antes de aplicar un GROUP BY.
- HAVING: Nos permite aplicar un filtro sobre los datos devueltos del GROUP BY (es equivalente al WHERE).
 - Se puede utilizar sin necesidad de utilizar un GROUP BY, en cuyo caso funciona de forma equivalente a un WHERE.
- CASE WHEN ---- ELSE --- END AS: Realizar acciones diferentes en función de una serie de condiciones impuestas.
 - Siempre debe terminar con END.
 - Se le puede poner alias de forma opcional. Aunque si no le ponemos alias no nos lo mostrara cuando no tengamos el CASE WHEN en un SELECT.
 - Podemos utilizarlo como campo solo en el SELECT, nos devolverá únicamente la columna con los valores especificados en las condiciones.
 - Podemos anidar CASE WHEN's uno dentro de otro. Aunque no suele ser recomendable.
 - Podemos utilizar CASE WHEN en:
 - SELECT
 - WHERE
 - HAVING
 - GROUP BY
 - Deberemos añadir la instrucción ELSE para los casos que no cumplan ninguna de las condiciones.

Pair programming Consultas avanzadas 2

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de pair programming. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind.

El día de hoy vamos a realizar ejercicios en los que practicaremos sentencias SQL usando el operador `GROUP BY`, que nos ayudará a agrupar o seleccionar los resultados según los valores de algunos de los atributos de la tabla, y `HAVING` y `CASE` para añadir condiciones a la ejecución de ciertas partes de las consultas de manera similar a como hacemos con `WHERE` (aunque con diferencias).

Ejercicios

1. Relación entre número de pedidos y máxima carga:

Desde logística nos piden el número de pedidos y la máxima cantidad de carga de entre los mismos (`freight`) que han sido enviados por cada empleado (mostrando el ID de empleado en cada caso).

2. Descartar pedidos sin fecha y ordénalos:

Una vez han revisado los datos de la consulta anterior, nos han pedido afinar un poco más el "disparo". En el resultado anterior se han incluido muchos pedidos cuya fecha de envío estaba vacía, por lo que tenemos que mejorar la consulta en este aspecto. También nos piden que ordenemos los resultados según el ID de empleado para que la visualización sea más sencilla.

3. Números de pedidos por día:

El siguiente paso en el análisis de los pedidos va a consistir en conocer mejor la distribución de los mismos según las fechas. Por lo tanto, tendremos que generar una consulta que nos saque el número de pedidos para cada día, mostrando de manera separada el día (`DAY()`), el mes (`MONTH()`) y el año (`YEAR()`).

4. Número de pedidos por mes y año:

La consulta anterior nos muestra el número de pedidos para cada día concreto, pero esto es demasiado detalle. Genera una modificación de la consulta anterior para que agrupe los pedidos por cada mes concreto de cada año.

5. Seleccionad las ciudades con 4 o más empleadas:

Desde recursos humanos nos piden seleccionar los nombres de las ciudades con 4 o más empleadas de cara a estudiar la apertura de nuevas oficinas.

6. Cread una nueva columna basándonos en la cantidad monetaria:

Necesitamos una consulta que clasifique los pedidos en dos categorías ("Alto" y "Bajo") en función de la cantidad monetaria total que han supuesto: por encima o por debajo de 2000 euros.

**Happy coding **

Consultas en múltiples tablas 1

Queries en múltiples tablas 1: CROSS JOIN, INNER JOIN, NATURAL JOIN

Producto Cartesiano (CROSS JOIN)

Las consultas a múltiples tablas pueden realizarse combinando las columnas y filas de las tablas de diferentes maneras. La manera más inmediata es mediante el producto cartesiano.

La sintaxis general para las consultas de tipo `CROSS JOIN` es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1  
FROM tabla1  
CROSS JOIN tabla2  
WHERE tabla1.columna1 = tabla2.columna1;
```

Una forma alternativa de escribirlas podría ser la siguiente, aunque suele ser menos usual:

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1  
FROM tabla1, tabla2  
WHERE tabla1.columna1 = tabla2.columna1;
```

Siempre recordando que las columnas de ambas tablas en el `WHERE`, deben ser iguales.

Dado que su funcionamiento puede resultar un poco confuso en ocasiones, vamos a verlo con un ejemplo práctico.

Volviendo a usar una versión de la tabla llamada `empleadas` de anteriores lecciones:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Ahora vamos a suponer que las empleadas trabajan en diferentes proyectos, por lo que tienen una tabla en la que recogen qué empleadas participan en qué proyectos. Esta tabla se llamará `empleadas_en_proyectos`:

id_empleada	id_proyecto
1	1
2	1
3	2
4	2
5	3

Imaginemos el caso concreto en el que la empresa quiere conocer los nombres y apellidos de cada empleada de la tabla empleadas y a que proyectos pertenece cada una. Para ello necesitaremos consultar la información de las dos tablas que hemos mencionado anteriormente, ya que los nombres están en una, mientras que la asignación de proyectos está en otra. Podremos realizar este tipo de consultas usando el producto cartesiano:

```
SELECT empleadas.nombre, empleadas.apellido, empleadas_en_proyectos.id_proyecto, ,empleadas_
FROM empleadas
CROSS JOIN empleadas_en_proyectos
WHERE empleadas.id_empleada = empleadas_en_proyectos.id_empleada;
```

En primer lugar, nos encontramos que dentro del `FROM` se indican (separadas con `,`) los nombres de las dos tablas sobre las que queremos realizar la consulta. De esta manera estamos haciendo saber al interprete de SQL que queremos realizar la consulta sobre el producto cartesiano de ambas tablas.

Ahora bien, ¿qué significa realizar un producto cartesiano de dos tablas? Simplemente se combinan todos los registros de la primera tabla con todos los registros de la segunda tabla:

nombre	apellido	id_empleada	id_proyecto
Ana	González	1	1
Maria	López	1	2
Lucia	Ramos	2	3
elena	Bueno	2	4
Rocío	García	3	5

NOTA: Por motivos de espacio se muestran solo las combinaciones para las 2 primeras empleadas de la tabla empleadas, pero sería similar para las restantes.

En la tabla-resultado anterior vemos como cada fila de la tabla empleadas se ha combinado con cada fila de la tabla `empleadas_en_proyectos` (tengan sentido realmente estas combinaciones o no). Por ejemplo, si la tabla empleadas tuviese 20 filas en total y la tabla `empleadas_en_proyectos` tuviese 20 filas también, la tabla-resultado al realizar el producto cartesiano de ambas tendría $20 \times 20 = 400$ filas, es decir, todas las posibles combinaciones de filas entre ambas tablas.

Si lo viéramos visualmente con colores, lo veríamos así:

Indice			CROSS JOIN		
1			Indice	1	Indice
2			1		3
			1		4
Indice			2		1
1			2		3
3			2		4
4			2		

CROSS

Sin embargo, muchas de las filas del producto cartesiano no tendrán sentido en este caso, siendo válidas solamente aquellas que comparten el mismo `id_empleada` para los registros de ambas tablas.

Para conseguir retirar las filas que no tienen sentido para nosotras en este caso hemos incluido una condición con el operador `WHERE`: quedarnos solamente con las filas en las que ambos atributos `id_empleada` sean iguales. Como ambas columnas a comparar se llamaban igual en las tablas originales, para indicar a qué tabla pertenece cada columna se debe indicar con el formato `nombre_tabla.nombre_columna`. Esto se realizará así siempre que usemos el producto cartesiano, se llamen igual las columnas o no.

Todo lo anterior nos dejaría con la siguiente tabla temporal sobre la que ejecutar finalmente el operador `SELECT`:

<code>id_empleada</code>	<code>salario</code>	<code>nombre</code>	<code>apellido</code>	<code>pais</code>	<code>id_empleada</code>	<code>id_proyec</code>
1	2500	Ana	González	España	1	1
2	4000	Maria	López	España	2	1
3	3000	Lucía	Ramos	España	3	2
4	5000	Elena	Bueno	España	4	2
5	1500	Rocío	García	Francia	5	3

Como podéis ver, se han dejado solo las columnas donde el `id_empleada` de ambas tablas es el mismo.

Finalmente, en el `SELECT` de nuestra consulta estamos indicando las columnas de cada tabla que queremos seleccionar para incluirlas en nuestro resultado. En este caso también se sigue el formato `nombre_tabla.nombre_columna`. Como hemos elegido quedarnos con `empleadas.nombre`, `empleadas.apellido` y `empleadas_en_proyectos.id_proyecto`, la tabla resultado final sería la siguiente:

nombre	apellido	id_proyecto
Ana	González	1
Maria	López	1
Lucía	Ramos	2
Elena	Bueno	2
Rocío	García	3

NOTA: Se pueden hacer consultas usando el producto cartesiano de más de 2 tablas. Sería el caso en el que además de tener la tabla `empleadas` y `empleadas_en_proyectos`, se tiene una tabla `Proyectos` con la información detallada de cada proyecto: `id_proyecto`, `nombre`, `descripción`, `presupuesto`, etc.

Ejercicio: Realiza la misma consulta, pero esta vez utilizando alias para las tablas involucradas.

Ejercicio: Selecciona los salarios medios de las empleadas asignadas al proyecto número 2 utilizando el producto cartesiano de las tablas `Empleada` y `empleadas_en_proyectos`.

NATURAL JOIN

El `NATURAL JOIN` es otro tipo de operador utilizado para realizar consultas a múltiples tablas. En este caso se diferencia del producto cartesiano "normal" o `CROSS JOIN` en que aquí se impone directamente la condición de que las columnas que se llaman igual en ambas tablas tengan el mismo valor, es decir, no hace falta el uso de `WHERE` para conseguirlo.

La sintaxis general para las consultas de tipo `NATURAL JOIN` es la siguiente:

```
SELECT columna1, columna2, columna3  
FROM tabla1 NATURAL JOIN tabla2;
```

Donde aunque no se especifique ambas tablas deben compartir una columna común, que se utiliza de forma interna para establecer las referencias entre ambas tablas.

Para comprender todo cómo funcionan las consultas con `NATURAL JOIN` vamos a replicar el ejemplo anterior en el que la empresa quiere conocer los nombres y apellidos de las empleadas y a qué proyectos pertenecen:

```
SELECT nombre, apellido, id_proyecto  
FROM empleadas NATURAL JOIN empleadas_en_proyectos;
```

En este caso se ha utilizado el operador `NATURAL JOIN`, por lo que no ha sido necesario el `WHERE` para indicar que queremos quedarnos solamente con aquellas entradas en las que las columnas `id_empleada` (que son las únicas que se llaman igual en ambas tablas) tengan el mismo valor. Por lo tanto, el resultado de la anterior consulta es el mismo que en el caso en el que usábamos el producto cartesiano junto con un `WHERE`:

nombre	apellido	id_proyecto
Ana	González	1
Maria	López	1
Lucía	Ramos	2
Elena	Bueno	2
Rocío	García	3

Es importante comprender que `NATURAL JOIN` elimina directamente las columnas duplicadas en ambas tablas, por lo que en este caso el resultado final solo contendría la columna `id_proyecto` una sola vez. Gracias a esto, no hemos tenido que especificar el nombre de la tabla dentro de la selección de columnas en el `SELECT`.

Ejercicio: Selecciona los salarios medios de las empleadas asignadas al proyecto número 2 realizando un `NATURAL JOIN` entre las tablas `Empleada` y `empleadas_en_proyectos`.

INNER JOIN

`INNER JOIN` es el caso más general de `NATURAL JOIN`, diferenciándose en que nos permite enlazar dos tablas usando columnas que no se llamen igual en ambas. Para ello se utiliza la palabra reservada `ON`:

-- `ON` se usa cuando las columnas a asociar no se llaman igual en ambas tablas

La sintaxis general para este tipo de consultas `INNER JOIN` es la siguiente:

```
SELECT columna1, columna2, tabla2.columna1
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

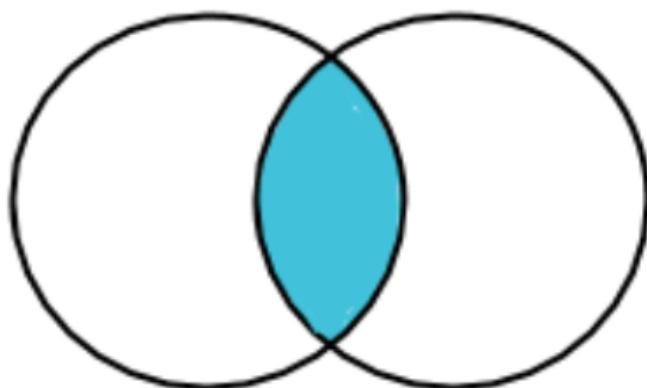
Donde en la parte de la instrucción `ON`, ambas columnas referenciadas (pueden llamarse con nombre distinto), contengan los mismos tipos de datos.

```
SELECT nombre, apellido, empleadas_en_proyectos.id_proyecto
FROM empleadas INNER JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada;
```

En la consulta anterior estamos de nuevo intentando seleccionar el nombre y apellido de las empleadas y el proyecto al que están asociadas. Si se diese el caso en el que la columna con el ID de las empleadas no se llamase de la misma manera en ambas tablas, usando `ON` podremos asignarlas manualmente. Podemos pensar en el funcionamiento de `ON` como si fuera el `WHERE` de un producto cartesiano.

Otra diferencia entre `NATURAL JOIN` e `INNER JOIN` está en que mientras que la primera devuelve una tabla-resultado en la que las columnas duplicadas se fusionan automáticamente, con `INNER JOIN` se devuelven por defecto todas las instancias de las columnas duplicadas (aunque se llamen igual), por lo que hay que hacer la selección manualmente. Es decir, el resultado de `INNER JOIN` contiene todas las columnas originales. Puede contener duplicados y nunca devolverá registros que sean `NULL` en ambas tablas.

Realmente podemos observar como `NATURAL JOIN` es un caso específico de `INNER JOIN` donde la cláusula `NATURAL` impone una serie de condiciones muy específicas al resultado de la consulta.



Inner Join

INNER JOIN: Diagrama de Venn

Si esto lo traducimos a lenguaje de tablas podríamos verlo de la siguiente forma:

Indice		
1		
2		

INNER JOIN

Indice				
1				

Indice		
1		
3		
4		

inner

Ejercicio: Selecciona los salarios medios de las empleadas asignadas al proyecto número 2 realizando un `INNER JOIN` entre las tablas `Empleada` y `empleadas_en_proyectos`.

Uso de INNER JOIN con el operador USING:

Si se quiere realizar un `INNER JOIN` en el que se igualen solamente dos filas concretas (como con `JOIN ON`) pero eliminando las columnas duplicadas de la misma manera que hacíamos en el caso de `NATURAL JOIN`, podemos emplear el operador `USING`:

```
SELECT nombre, apellido, empleadas_en_proyectos.id_proyecto  
FROM empleadas INNER JOIN empleadas_en_proyectos  
USING (id_empleada);
```

En este caso, hemos realizado el `JOIN` (cuando no se indica el tipo de `JOIN` el intérprete asume que es `INNER JOIN`) tomando como referencia común para ambas tablas la columna `id_empleada`, que se llama igual en ambas tablas.

La tabla que obtendremos como resultado de unir `empleadas` y `empleadas_en_proyecto` usando `USING` solo contendrá una copia de la columna indicada en el `USING` (a diferencia de `JOIN ON` que sí contendrá por duplicado las columnas asignadas). Si trabajamos con MySQL, es importante que la/las columnas que indiquemos dentro del `USING` estén entre paréntesis, aunque solo sea una.

NOTA: `ON` y `USING` son excluyentes entre sí ya que son dos maneras de asignar las columnas por las que se realizará el `JOIN`, y entrarían en conflicto.

Ejercicio: Selecciona los salarios medios de las empleadas asignadas al proyecto número 2 realizando un `INNER JOIN` con `USING` entre las tablas `empleadas` y `empleadas_en_proyectos`.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisa la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla `customers` tiene las siguientes columnas:

- `customer_number` : el número identificativo de las clientas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientas/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita mas espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

La tabla Employees tiene las siguientes columnas:

- `employee_number` : el número identificativo de las empleadas/os. Es un número entero y sirve de clave primaria.
- `last_name` : el apellido de las empleadas. Es una cadena de texto.
- `first_name` : el nombre de las empleadas. Es una cadena de texto.
- `extension` : su extensión telefónica. Es una cadena de texto.
- `email` : el correo electrónico de la empleada. Es una cadena de texto.
- `office_code` : El código de la oficina de la empleada. Es una cadena de texto.
- `reports_to` : el número identificativo de la empleada a la que reporta (su supervisora). Es un número entero y clave foránea (relacionada con employeeNumber).
- `job_title` : el nombre del puesto de trabajo que desempeña. Es una cadena de texto.

EJERCICIO 1

Selecciona el ID, nombre, apellidos de las empleadas y el ID de cada cliente asociado a ellas, usando `CROSS JOIN`.

EJERCICIO 2

Selecciona el ID, nombre, apellidos de las empleadas, para aquellas con más de 8 clientes, usando `CROSS JOIN`.

EJERCICIO 3

Selecciona el nombre y apellidos de las empleadas que tienen clientes de más de un país, usando `CROSS JOIN`.

EJERCICIO 4

Selecciona el ID, nombre, apellidos de las empleadas y el ID de cada cliente asociado a ellas, usando `INNER JOIN`.

EJERCICIO 5

Selecciona el ID, nombre, apellidos de las empleadas, para aquellas con más de 8 clientes, usando `INNER JOIN`.

EJERCICIO 6

Selecciona el nombre y apellidos de las empleadas que tienen clientes de más de un país, usando `INNER JOIN`.

Repaso Consultas en Múltiples tablas 1

- Es buena práctica indicar de que tabla contiene el atributo(columna) que queremos seleccionar en el SELECT.
 - eg:

```
SELECT tabla1.columna1 , tabla.columna2 ....
FROM tabla1
NATURAL JOIN
tabla2;
```
- CROSS JOIN: Selecciona todas las posibles combinaciones de los registros de ambas tablas (aplica el producto cartesino).
 - Si utilizamos el WHERE indicando las columnas de union nos eliminará los posibles duplicados.
 - Las columnas que encontramos en el WHERE deben de ser iguales.
- NATURAL JOIN: Es un caso particular de INNER JOIN en el cual no es necesario indicar la columnas por las cuales queremos hacer las uniones de ambas tablas, ya que en ambas tiene el mismo nombre.
 - No es necesario utilizar la clausula ON para que haga la operación de union de tablas.
 - Nos devuelve los resultados sin duplicados.
- INNER JOIN: Nos devuelve la intersección de ambas tablas, es decir, únicamente nos devuelve los registros comunes a ambas tablas.
 - El ON nos especifica la columna común a ambas tablas. Por lo tanto, podemos referenciarlos cuando el no tenga la columna de union de las tablas el mismo nombre en ambas.
 - No elimina los posibles duplicados que puedan existir al hacer la consulta en ambas tablas.
 - USING: Nos permite eliminar duplicados siempre que la columna utilizada en el USING tenga el mismo nombre en ambas tablas. (Podemos replicar un NATURAL JOIN con un INNER JOIN utilizando el USING).
 - Nos permite filtrar mediante funciones agregadas sobre los datos un INNER JOIN siempre y cuando apliquemos previamente un GROUP BY.
 - USING y ON son excluyentes, deberemos utilizar uno o el otro.

Pair programming Consultas en múltiples tablas

1

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de pair programming. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos `Northwind`.

El día de hoy vamos a realizar ejercicios en los que practicaremos las queries SQL a múltiples tablas usando los operadores `CROSS JOIN`, `INNER JOIN` y `NATURAL JOIN`. De esta manera podremos combinar los datos de diferentes tablas en las mismas bases de datos, para así realizar consultas mucho mas complejas.

Ejercicios

1. Pedidos por empresa en UK:

Desde las oficinas en UK nos han pedido con urgencia que realicemos una consulta a la base de datos con la que podamos conocer cuántos pedidos ha realizado cada empresa cliente de UK. Nos piden el ID del cliente y el nombre de la empresa y el número de pedidos.

Deberéis obtener una tabla similar a esta:

	NombreEmpresa	Identificador	NumeroPedidos
▶	Around the Horn	AROUT	13
	B's Beverages	BSBEV	10
	Consolidated Holdings	CONSH	3
	Eastern Connection	EASTC	8
	Island Trading	ISLAT	10
	North/South	NORTS	3
	Seven Seas Imports	SEVES	9

sql61

2. Productos pedidos por empresa en UK por año:

Desde Reino Unido se quedaron muy contentas con nuestra rápida respuesta a su petición anterior y han decidido pedirnos una serie de consultas adicionales. La primera de ellas consiste en una *query* que nos sirva para conocer cuántos objetos ha pedido cada empresa cliente de UK durante cada año. Nos piden concretamente conocer el nombre de la empresa, el año, y la cantidad de objetos que han pedido. Para ello hará falta hacer 2 joins.

El resultado será una tabla similar a esta:

	NombreEmpresa	Año	NumObjetos
▶	Around the Horn	1996	105
	Around the Horn	1997	371
	Around the Horn	1998	174
	B's Beverages	1996	39
	B's Beverages	1997	165
	B's Beverages	1998	89
	Consolidated Holdings	1997	54
	Consolidated Holdings	1998	33
	Eastern Connection	1996	35
	Eastern Connection	1997	155
	Eastern Connection	1998	379
	Island Trading	1996	80
	Island Trading	1997	122

sql62

3. Mejorad la *query* anterior:

Lo siguiente que nos han pedido es la misma consulta anterior pero con la adición de la cantidad de dinero que han pedido por esa cantidad de objetos, teniendo en cuenta los descuentos, etc. Ojo que los descuentos en nuestra tabla nos salen en porcentajes, 15% nos sale como `0 . 15`.

La tabla resultante será:

	NombreEmpresa	Año	NumObjetos	DineroTotal
▶	Around the Horn	1996	105	1379
	Around the Horn	1997	371	6406.8999958
	Around the Horn	1998	174	5604.75
	B's Beverages	1996	39	479.4
	B's Beverages	1997	165	3179.5
	B's Beverages	1998	89	2431
	Consolidated Holdings	1997	54	787.6
	Consolidated Holdings	1998	33	931.5
	Eastern Connection	1996	35	950
	Eastern Connection	1997	155	4514.35
	Eastern Connection	1998	379	9296.685000...
	Island Trading	1996	80	901.1999999...
	Island Trading	1997	122	2560.5

sql63

4. **BONUS:** Pedidos que han realizado cada compañía y su fecha:
 Después de estas solicitudes desde UK y gracias a la utilidad de los resultados que se han obtenido, desde la central nos han pedido una consulta que indique el nombre de cada compañía cliente junto con cada pedido que han realizado y su fecha.

El resultado deberá ser:

	OrderID	CompanyName	OrderDate
▶	10643	Alfreds Futterkiste	1997-08-25 00:00:00
	10692	Alfreds Futterkiste	1997-10-03 00:00:00
	10702	Alfreds Futterkiste	1997-10-13 00:00:00
	10835	Alfreds Futterkiste	1998-01-15 00:00:00
	10952	Alfreds Futterkiste	1998-03-16 00:00:00
	11011	Alfreds Futterkiste	1998-04-09 00:00:00
	10308	Ana Trujillo Emparedados y helados	1996-09-18 00:00:00
	10625	Ana Trujillo Emparedados y helados	1997-08-08 00:00:00
	10759	Ana Trujillo Emparedados y helados	1997-11-28 00:00:00
	10926	Ana Trujillo Emparedados y helados	1998-03-04 00:00:00
	10365	Antonio Moreno Taquería	1996-11-27 00:00:00
	10507	Antonio Moreno Taquería	1997-04-15 00:00:00
	10535	Antonio Moreno Taquería	1997-05-13 00:00:00

sql64

5. **BONUS:** Tipos de producto vendidos:

Ahora nos piden una lista con cada tipo de producto que se han vendido, sus categorías, nombre de la categoría y el nombre del producto, y el total de dinero por el que se ha vendido cada tipo de producto (teniendo en cuenta los descuentos).

Pista Necesitaréis usar 3 joins.

	CategoryID	CategoryName	ProductName	ProductSales
▶	1	Beverages	Chai	12788.1
	1	Beverages	Chang	16355.96
	1	Beverages	Chartreuse verte	12294.539999999999
	1	Beverages	Côte de Blaye	141396.74
	1	Beverages	Guaraná Fantástica	4504.36
	1	Beverages	Ipoh Coffee	23526.699999999997
	1	Beverages	Lakkaliköri	15760.44
	1	Beverages	Laughing Lumberjack Lager	2396.8
	1	Beverages	Outback Lager	10672.65
	1	Beverages	Rhönbräu Klosterbier	8177.479999999999
	1	Beverages	Sasquatch Ale	6350.4
	1	Beverages	Steeleye Stout	13643.999999999998
	2	Condiments	Aniseed Syrup	3044
	2	Condiments	Chef Anton's Cajun Seas...	8567.900000000001
	2	Condiments	Chef Anton's Gumbo Mix	5347.21

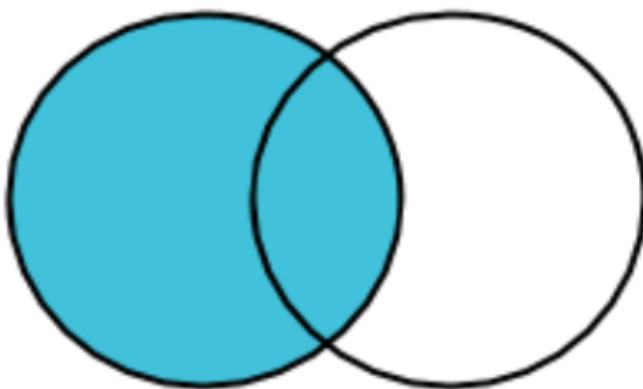
sql65

Happy coding

Consultas en múltiples tablas 2

En la lección anterior comenzamos a ver cómo realizar consultas combinando datos de múltiples tablas. Sin embargo, existen maneras diferentes de hacer la combinación de las tablas en la consulta. Con `INNER` y `NATURAL JOIN` vimos como realizar una combinación más o menos "estándar" en la que se daba el mismo tratamiento a ambas tablas. Sin embargo, existirán casos más especiales o específicos en los que no queramos tratar a ambas tablas de la misma manera. Es en este punto cuando el uso de operadores como `LEFT` y `RIGHT JOIN` se vuelve interesante.

LEFT JOIN



Left Join

LEFT JOIN: Diagrama de Venn

Como podemos observar en el diagrama de Venn superior, una consulta SQL a múltiples tablas creada usando `LEFT JOIN` seleccionará todos los registros de la tabla nombrada en primer lugar (la situada la izquierda del `JOIN`), mientras que de la tabla nombrada en segundo lugar solo tendrá en cuenta los registros coincidentes, es decir los que realizan intersección con la primera tabla. El resto de registros de la tabla derecha se descartarán. Como en el caso de `INNER JOIN`, la asociación entre las columnas de ambas tablas se realizará usando el operador `ON`.

Por otro lado, en el caso de los registros de la tabla izquierda que no tienen valores asociados en la tabla derecha, los atributos/columnas provenientes de la tabla derecha tomarán el valor `NULL` en el resultado.

Realicemos un ejemplo para ver el comportamiento real de `LEFT JOIN`. Supongamos que ahora las tablas `empleadas` y `empleadas_en_proyectos` son las siguientes:

id_empleada	salario	nombre	apellido	país
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
5	1500	Rocío	García	Francia

id_empleada	id_proyecto
2	1
3	2
4	2
5	3

La sintaxis general para las consultas de tipo `LEFT JOIN` es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1, ....
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

Donde de nuevo en la parte de la instrucción `ON`, se tiene que referenciar dos columnas iguales que existan en ambas tablas, por ejemplo algún ID.

Si quisiéramos conocer los nombres y apellidos de todas las empleadas y sus proyectos asociados (si es que los tuviesen) podríamos realizar la siguiente consulta usando `LEFT JOIN`:

```
SELECT empleadas.nombre, empleadas.apellido, empleadas_en_proyectos.id_proyecto
FROM empleadas LEFT JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada;
```

En este caso, la tabla izquierda del `JOIN` sería `empleadas`, de la cual se devolverían todos los registros. Realmente esto es lo que buscamos al usar `LEFT JOIN` ya que queremos conocer el nombre y apellidos de *todas* las empleadas (estén asignadas a un proyecto o no).

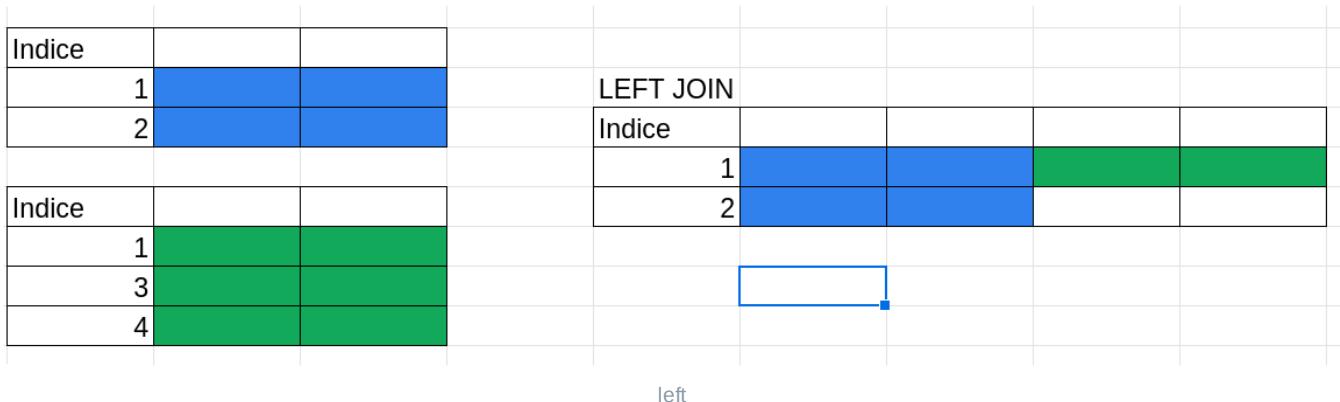
Por su parte, la tabla derecha del `JOIN` sería `empleadas_en_proyectos`, de la cual se devuelven únicamente los registros que estén asociados a la primera tabla. Usando el operador `ON` se ha indicado al intérprete SQL que la columna que se debe usar para buscar dichas asociaciones sea `id_empleada`. De nuevo, esto se ajusta a lo que buscamos conseguir en la consulta ya que queremos seleccionar los `id_proyecto` solamente de aquellas empleadas que existan en la tabla `empleadas`.

El resultado final de la consulta sería el siguiente:

nombre	apellido	id_proyecto
Ana	González	NULL
Maria	López	1
Lucía	Ramos	2
Rocío	García	3

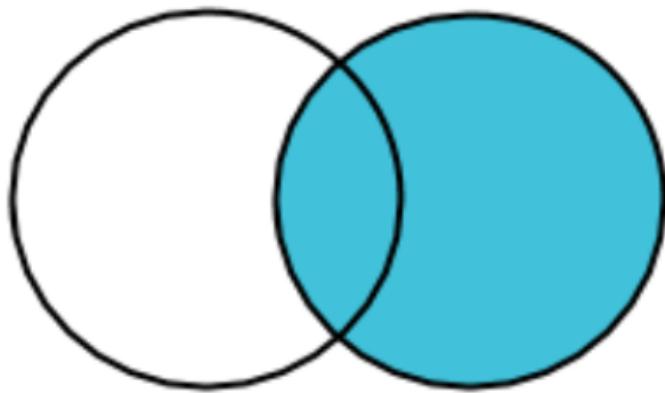
En la tabla anterior podemos ver también otra de las particularidades de `LEFT JOIN` que hemos indicado en su definición, que es el hecho de que los registros de la primera tabla que no tienen valores asociados para las columnas provenientes de la segunda tabla se completan con el valor `NULL`. Por ejemplo, como la empleada con `id_empleada = 1`, es decir, Ana González, no tiene ninguna entrada asociada en la tabla `empleadas_en_proyectos` (un registro en el que `id_empleada` valga 1 también), el valor que tomará la columna `id_proyecto` en la tabla-resultado es `NULL`. Todas las demás entradas sí tienen valor coincidente.

¿Qué es lo que está ocurriendo si lo vemos con colores?



La principal diferencia con `INNER JOIN` es mientras que `LEFT JOIN` incorpora todos los registros de la primera tabla aunque no tengan entrada asociada en la segunda tabla, `INNER JOIN` solo selecciona aquellos registros con valores asociados en ambas tablas. Volviendo de nuevo al ejemplo anterior, la fila correspondiente a Ana González no se seleccionaría / incluiría si hubiésemos realizado un `INNER JOIN`.

RIGHT JOIN



Right Join

RIGHT JOIN: Diagrama de Venn

En este caso (como puede verse en el diagrama de Venn superior), el funcionamiento de `RIGHT JOIN` es similar al de `LEFT JOIN`, pero cambiando el orden de prioridad de las tablas, devolviendo siempre todos los registros de la tabla de la derecha del `JOIN`, mientras que de la tabla a la izquierda se seleccionan solo aquellas entradas con registro asociado en la tabla derecha (su intersección).

La sintaxis general para las consultas de tipo `RIGHT JOIN` es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1, ...
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;
```

Donde de nuevo en la parte de la instrucción `ON`, se tiene que referenciar dos columnas iguales que existan en ambas tablas, por ejemplo algún ID.

Veámoslo con un ejemplo:

```
SELECT empleadas.nombre, empleadas.apellido, empleadas_en_proyectos.id_proyecto
FROM empleadas RIGHT JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada;
```

La consulta superior tiene exactamente la misma sintaxis que en el ejemplo del uso de `LEFT JOIN`, con el único cambio de la cláusula `LEFT` por `RIGHT`. Debido a ese cambio podemos ver cómo el resultado que obtenemos es bastante diferente:

nombre	apellido	id_proyecto
Maria	López	1
Lucía	Ramos	2
Rocío	García	3
NULL	NULL	2

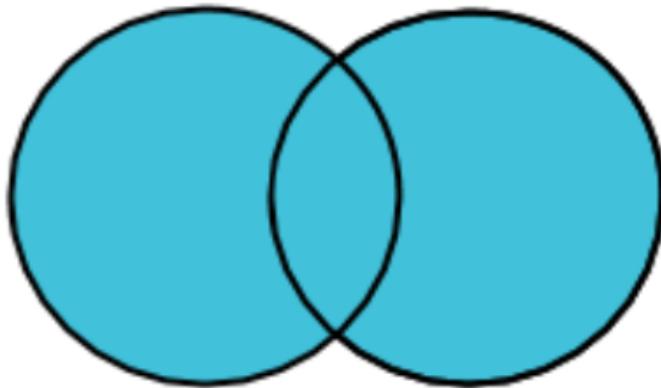
Debido al uso de `RIGHT JOIN`, en este caso *todos* los registros de `empleadas_en_proyectos` se incluyen en la tabla-resultado. Sus atributos/columnas se combinarán con los de los registros coincidentes de la tabla `empleadas` usando el atributo `id_empleada` como referencia. Ya que para algunos registros de `empleadas_en_proyectos` como el tercero de ellos (`id_empleada = 4` e `id_proyecto = 2`) no existe coincidencia en la primera tabla, los atributos correspondientes (nombre y apellido) tomarán el valor `NULL`.

¿Y cómo lo veríamos con colores?



FULL (OUTER) JOIN

NOTA: No hay que darle importancia a este tipo de consulta, ya que es inusual su uso. Pero es importante conocerla. No os demoréis mucho en entenderla o hacer los ejercicios con todo lujo de detalles.



Full Outer Join

FULL JOIN: Diagrama de Venn

Puede que encuentres un `FULL JOIN`, un `OUTER JOIN`, o un `FULL OUTER JOIN`. En la imagen aquí arriba podemos observar que estas consultas devuelven todos los registros de ambas tablas (tengan entradas asociadas en la otra tabla o no). En este caso, la única condición para que se lleve a cabo la combinación de las tablas es que exista *al menos una* coincidencia de registros entre las dos tablas. Con que haya al menos uno sería suficiente. De no haber ninguna, la tabla resultado estaría vacía. De la misma manera que en `LEFT` y `RIGHT JOIN`, las columnas provenientes de una de las tablas que no tengan un registro asociado en la otra tabla tomarán el valor `NULL`.

Una vez más, si nos fijamos en el diagrama superior podemos observar que las consultas llevadas a cabo utilizando `OUTER JOIN` devuelven todos los registros de ambas tablas (tengan entradas asociadas en la otra tabla o no). En este caso, la única condición para que se lleve a cabo la combinación de las tablas es que exista *al menos una* coincidencia de registros entre las dos tablas. Con que haya al menos uno sería suficiente. De no haber ninguna, la tabla resultado estaría vacía. De la misma manera que en `LEFT` y `RIGHT JOIN`, las columnas provenientes de una de las tablas que no tengan un registro asociado en la otra tabla tomarán el valor `NULL`.

Por desgracia, en MySQL no existe la instrucción `FULL (OUTER) JOIN`, de forma que para replicarla hemos de escribir dicha consulta haciendo por un lado un `LEFT JOIN`, junto con una instrucción que explicaremos en la siguiente clase llamada `UNION` y tras esto realizar un `RIGHT JOIN`. Escribiéndolo de esta forma replicamos el uso de un `FULL JOIN`, si dicha instrucción existiera en MySQL.

Dicho de otra modo, podemos pensar en `FULL (OUTER) JOIN` como una union de los resultados de una consulta `LEFT JOIN` con los resultados de la misma consulta pero con `RIGHT JOIN`, eliminando posteriormente las filas duplicadas.

La sintaxis general para las consultas de tipo `FULL (OUTER) JOIN` es la siguiente:

```

SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1...
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1
UNION
SELECT tabla1.columna1, tabla1.columna2, tabla2.columna1...
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna1 = tabla2.columna1;

```

Donde de nuevo en la parte de la instrucción `ON`, se tiene que referenciar dos columnas iguales que existan en ambas tablas, por ejemplo algún ID.

Ejemplo: Seleccionamos los nombres y apellidos de todas las empleadas de la tabla `empleadas`, así como los `id_proyecto` de todos los proyectos de la tabla `empleadas_en_proyectos`. Si alguna de las empleadas está asignada a un proyecto, se indicará en el resultado:

```

SELECT empleadas.nombre, empleadas.apellido, empleadas_en_proyectos.id_proyecto
FROM empleadas
LEFT JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada
UNION
SELECT empleadas.nombre, empleadas.apellido, empleadas_en_proyectos.id_proyecto
FROM empleadas
RIGHT JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada;

```

Una vez más seguimos la misma sintaxis que en los otros tipos de `JOIN`, en la que consultamos al mismo tiempo las tablas `empleadas` y `empleadas_en_proyectos`. El resultado al usar `FULL (OUTER) JOIN` es el siguiente (y diferente al que obtuvimos al usar `LEFT` y `RIGHT JOIN`):

nombre	apellido	id_proyecto
Ana	González	NULL
Maria	López	1
Lucía	Ramos	2
Rocío	García	3
NULL	NULL	2

Debido a que hemos realizado un `FULL (OUTER) JOIN`, los resultados contienen todos los registros existentes de ambas tablas. Aquellos atributos para los que no existen coincidencias en alguna de las tablas tomarán el valor `NULL` (la columna `id_proyecto` para Ana González por ejemplo).

De nuevo, con colores lo veríamos así:

Indice			OUTER JOIN		
1			1		
2			2		
Indice					
1			1		
3			2		
4			3		
			4		
full					

NOTAS:

- Igual que cuando trabajamos con productos cartesianos (CROSS JOIN), con FULL JOIN también hay que tener cuidado al combinar tablas con un gran número de registros, ya que los resultados potenciales serán muy grandes.
- FULL JOIN, FULL (OUTER) JOIN y OUTER JOIN son el mismo operador. Son distintas maneras de denominarlo, el nombre concreto a utilizar dependerá del SGBD.

SELF JOIN

SELF JOIN es un caso concreto de NATURAL JOIN en el que una tabla se combina consigo misma en vez de con una segunda tabla. Debido a esta condición especial, se deben usar aliases para cada instancia de la tabla, de cara a distinguir entre las columnas de una o de otra (ya que se llaman igual por defecto). Veámoslo con un ejemplo en el que realizamos un SELF JOIN de la tabla empleadas consigo misma:

```
SELECT A.nombre AS Nombre1, A.apellido AS Apellido1, A.salario AS Salario1, B.nombre AS Nombre2, B.apellido AS Apellido2, B.salario AS Salario2
FROM empleadas AS A, empleadas AS B
WHERE A.id_empleada <> B.id_empleada
AND A.pais = B.pais;
```

Como hemos indicado anteriormente, para poder diferenciar entre las dos versiones de la tabla empleadas se le asigna un alias a cada una (A y B en este caso). El objetivo de la consulta es seleccionar parejas de empleadas del mismo país y visualizar sus salarios, lo cual puede resultar útil para encontrar diferencias significativas entre ellos e investigar la causa.

Como en el caso de NATURAL JOIN, la combinación de tablas no necesita operador ON, y las condiciones para crear las parejas de empleadas (mismo país y descartar las comparaciones entre una empleada y ella misma), es decir, para combinar registros de cada tabla también se imponen con el uso del WHERE.

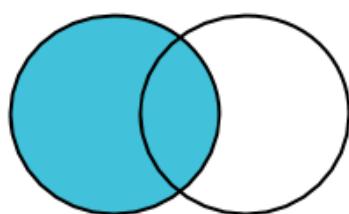
La tabla Resultado sería la siguiente:

Nombre1	Apellido1	Salario1	Nombre2	Apellido2	Salario2	pais
Ana	González	2500	Maria	López	4000	España
Ana	González	2500	Lucía	Ramos	3000	España
Maria	López	3000	Ana	González	2500	España
Maria	López	1500	Lucía	Ramos	4000	España
Lucía	Ramos	4000	Ana	González	2500	España
Lucía	Ramos	4000	Maria	López	3000	España

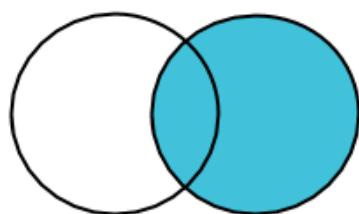
Podemos ver como los registros correspondientes a la empleada Rocío García no se han incluido en el resultado ya que es la única empleada en Francia y por lo tanto no tiene otras empleadas con las que comparar su salario.

Por otro lado, podemos ver registros duplicados en la tablaResultado, ya que las parejas comparadas se repiten en orden inverso. Debido a ello hay que tener cuidado al usar `SELF JOIN` y pensar bien las condiciones en `WHERE` para que los resultados no alcancen un tamaño demasiado grande.

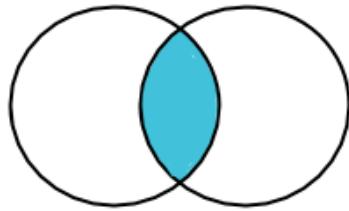
Resumen JOINs



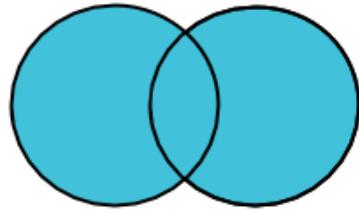
Left Join



Right Join



Inner Join



Full Outer Join

El diagrama de Venn superior resume los tipos de `JOIN` entre dos tablas diferentes que hemos estudiado en las dos últimas lecciones. En él se puede apreciar la diferencia de dos grandes categorías: aquellos `JOIN`s que sólo incluyen tuplas de registros que se corresponden entre las dos tablas (`INNER JOIN`) y por otro lado, los `JOIN`s que además de las coincidencias incluyen también todos los registros de una o de ambas tablas (como `LEFT`, `RIGHT` y `FULL (OUTER) JOIN`).

NOTA: De la misma manera que `NATURAL JOIN` era un caso más restringido de `INNER JOIN` donde la cláusula `NATURAL` imponía ciertas condiciones, a cualquiera de los otros tipos de `JOIN` también se les puede añadir el operador `NATURAL` delante, lo que hará que se le impongan las mismas restricciones. A saber: que se comparan las columnas con el mismo nombre en ambas tablas sin la necesidad de la cláusula `ON` y que se eliminen las columnas duplicadas, ambas cosas de manera automática.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Si no la tienes o si tienes dudas de como importarla, revisa la página asociada de [tutorial](#). Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla `customers` tiene las siguientes columnas:

- `customer_number` : el número identificativo de las cuentas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientes/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita más espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

La tabla Employees tiene las siguientes columnas:

- `employee_number` : el número identificativo de las empleadas/os. Es un número entero y sirve de clave primaria.
- `last_name` : el apellido de las empleadas. Es una cadena de texto.
- `first_name` : el nombre de las empleadas. Es una cadena de texto.
- `extension` : su extensión telefónica. Es una cadena de texto.
- `email` : el correo electrónico de la empleada. Es una cadena de texto.
- `office_code` : El código de la oficina de la empleada. Es una cadena de texto.
- `reports_to` : el número identificativo de la empleada a la que reporta (su supervisora). Es un número entero y clave foránea (relacionada con employeeNumber).
- `job_title` : el nombre del puesto de trabajo que desempeña. Es una cadena de texto.

EJERCICIO 1

Selecciona el ID, nombre, apellidos de todas las empleadas y el ID de cada cliente asociado a ellas (si es que lo tienen).

EJERCICIO 2

Selecciona el ID de todos los clientes, y el nombre, apellidos de las empleadas que llevan sus pedidos (si es que las hay).

EJERCICIO 3

Selecciona el ID, nombre, apellidos de las empleadas, para aquellas con más de 8 clientes, usando `LEFT JOIN`.

EJERCICIO 4

Selecciona el ID, nombre, apellidos de las empleadas, para aquellas con más de 8 clientes, usando `RIGHT JOIN`.

EJERCICIO 5

Selecciona el nombre y apellidos de las empleadas que tienen clientes de más de un país, usando `LEFT JOIN`.

Pair programming Consultas en múltiples tablas

2

Nota: veréis que os hemos propuesto bastantes ejercicios para cada sesión de pair programming. No hace falta hacerlos todos. Estos ejercicios están pensados para que cada día podáis hacer los primeros ejercicios aunque no hayáis terminado los últimos del día anterior. Cada día debéis hacer los ejercicios de este día, no los que no terminasteis el día anterior.

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind.

El día de hoy vamos a realizar ejercicios en los que practicaremos las queries SQL a múltiples tablas usando los operadores `LEFT JOIN` , `RIGHT JOIN` , `SELF JOIN` . De esta manera podremos combinar los datos de diferentes tablas en las mismas bases de datos, para así realizar consultas mucho mas complejas.

Ejercicios

1. Qué empresas tenemos en la BBDD Northwind:

Lo primero que queremos hacer es obtener una consulta SQL que nos devuelva el nombre de todas las empresas cliente, los ID de sus pedidos y las fechas.

Los resultados deberán similares a la siguiente tabla:

	OrderID	CompanyName	OrderDate
▶	10643	Alfreds Futterkiste	1997-08-25 00:00:00
	10692	Alfreds Futterkiste	1997-10-03 00:00:00
	10702	Alfreds Futterkiste	1997-10-13 00:00:00
	10835	Alfreds Futterkiste	1998-01-15 00:00:00
	10952	Alfreds Futterkiste	1998-03-16 00:00:00
	11011	Alfreds Futterkiste	1998-04-09 00:00:00
	10308	Ana Trujillo Emparedados y helados	1996-09-18 00:00:00
	10625	Ana Trujillo Emparedados y helados	1997-08-08 00:00:00
	10759	Ana Trujillo Emparedados y helados	1997-11-28 00:00:00
	10926	Ana Trujillo Emparedados y helados	1998-03-04 00:00:00
	10365	Antonio Moreno Taquería	1996-11-27 00:00:00
	10507	Antonio Moreno Taquería	1997-04-15 00:00:00
	10535	Antonio Moreno Taquería	1997-05-13 00:00:00
	10573	Antonio Moreno Taquería	1997-06-19 00:00:00
	10677	Austria Matten	1997-09-22 00:00:00

sql71

2. Pedidos por cliente de UK:

Desde la oficina de Reino Unido (UK) nos solicitan información acerca del número de pedidos que ha realizado cada cliente del propio Reino Unido de cara a conocerlos mejor y poder adaptarse al mercado actual. Específicamente nos piden el nombre de cada compañía cliente junto con el número de pedidos.

La tabla resultante será:

	NombreCliente	NumeroPedidos
▶	Around the Horn	13
	B's Beverages	10
	Consolidated Holdings	3
	Eastern Connection	8
	Island Trading	10
	North/South	3
	Seven Seas Imports	9

sql72

3. Empresas de UK y sus pedidos:

También nos han pedido que obtengamos todos los nombres de las empresas cliente de Reino Unido (tengan pedidos o no) junto con los ID de todos los pedidos que han realizado, el nombre de contacto de cada empresa y la fecha del pedido.

Los resultados de la query deberán ser:

	OrderID	NombreCliente	FechaPedido
▶	10355	Around the Horn	1996-11-15 00:00:00
	10383	Around the Horn	1996-12-16 00:00:00
	10453	Around the Horn	1997-02-21 00:00:00
	10558	Around the Horn	1997-06-04 00:00:00
	10707	Around the Horn	1997-10-16 00:00:00
	10741	Around the Horn	1997-11-14 00:00:00
	10743	Around the Horn	1997-11-17 00:00:00
	10768	Around the Horn	1997-12-08 00:00:00
	10793	Around the Horn	1997-12-24 00:00:00

sql73

4. Empleadas que sean de la misma ciudad:

Ejercicio de **SELF JOIN** : Desde recursos humanos nos piden realizar una consulta que muestre por

pantalla los datos de todas las empleadas y sus supervisoras. Concretamente nos piden: la ubicación, nombre, y apellido tanto de las empleadas como de las jefas. Investiga el resultado, ¿sabes decir quién es el director?

La tabla resultado de la query deberá ser:

	city	NombreEmpleado	ApellidoEmpleado	city	NombreJefe	ApellidoJefe
▶	Kirkland	Janet	Leverling	Tacoma	Andrew	Fuller
	London	Anne	Dodsworth	London	Steven	Buchanan
	London	Robert	King	London	Steven	Buchanan
	London	Michael	Suyama	London	Steven	Buchanan
	London	Steven	Buchanan	Tacoma	Andrew	Fuller
	Redmond	Margaret	Peacock	Tacoma	Andrew	Fuller
	Seattle	Laura	Callahan	Tacoma	Andrew	Fuller
	Seattle	Nancy	Davolio	Tacoma	Andrew	Fuller

sql74

ESTE EJERCICIO NO SE EVALUARÁ SI NO ES ENTREGADO

1. BONUS: FULL OUTER JOIN Pedidos y empresas con pedidos asociados o no:

Selecciona todos los pedidos, tengan empresa asociada o no, y todas las empresas tengan pedidos asociados o no. Muestra el ID del pedido, el nombre de la empresa y la fecha del pedido (si existe). La tabala resultado deberá similar a:

	OrderID	NombreCliente	FechaPedido
▶	10355	Around the Horn	1996-11-15 00:00:00
	10383	Around the Horn	1996-12-16 00:00:00
	10453	Around the Horn	1997-02-21 00:00:00
	10558	Around the Horn	1997-06-04 00:00:00
	10707	Around the Horn	1997-10-16 00:00:00
	10741	Around the Horn	1997-11-14 00:00:00
	10743	Around the Horn	1997-11-17 00:00:00
	10768	Around the Horn	1997-12-08 00:00:00
	10793	Around the Horn	1997-12-24 00:00:00

sql75

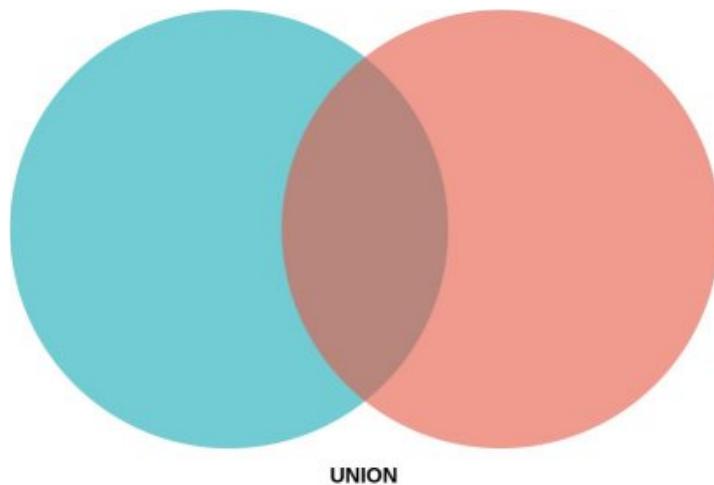
Happy coding

Consultas en múltiples tablas 3

Consultas en múltiples tablas 3: UNION, UNION ALL, INTERSECT, EXCEPT

En las lecciones anteriores vimos como utilizar algunas instrucciones de SQL para poder realizar consultas en diferentes tablas, de formas diversas, en función del tipo de resultados que estábamos buscando. La particularidad de este tipo de instrucciones es que nos permite combinar dos instrucciones de tipo `SELECT`. Adicionalmente existen otros métodos que explicaremos a continuación.

UNION



UNION Diagrama de Venn

Como se puede ver en el diagrama de Venn, de la imagen superior, la instrucción UNION se utiliza para combinar los resultados de dos o más instrucciones del tipo `SELECT`, con la particularidad de que los resultados siempre se devuelven sin filas duplicadas para las tablas utilizadas. Se han de tener en cuenta algunas consideraciones a la hora de poder utilizar la instrucción UNION, ya que si no, nos dará error y no se podrá utilizar.

- Cada instrucción de tipo `SELECT` en la UNION debe tener el mismo número de columnas
- Las columnas deben tener tipos de datos iguales (int, str, etc)

Supongamos que ahora las tablas clientes y proyectos de nuestro ejemplo son las siguientes:

id_clientes	id_proyecto	nombre	ciudad	pais
1	1	HR Analitycs ES	Madrid	España
2	2	Luxury Brands	Paris	Francia
3	3	Preservación Bosques	Lisboa	Portugal
4	4	Data Inc.	Berlín	Alemania
5	5	Data Italia	Roma	Italia

id_proyecto	nombre	ciudad	pais
1	Predicción Salarios	Salamanca	España
2	Agrupaciones Marcas	Brest	Francia
3	Visualización Bosques	Manteigas	Portugal
4	Corrección de datos	Berlín	Alemania
5	Creación de data lake	Nápoles	Italia

Vamos ahora a ver un ejemplo de la sentencia `UNION` con las tablas anteriores:

```
SELECT pais
FROM clientes
UNION
SELECT pais
FROM proyectos;
```

Al ejecutar la sentencia anterior nos devolverá una lista con los valores únicos de las ciudades contenidas en ambas tablas, de esta forma, el resultado sería:

pais
España
Francia
Portugal
Alemania
Italia

Fíjate que los países únicamente aparecen una vez, esto se debe a que `UNION` nunca muestra aquellos valores duplicados.

Visualmente sería así:

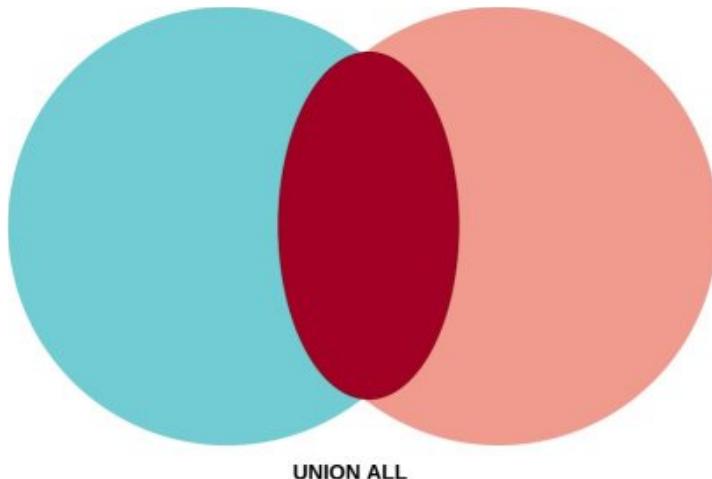
Indice		
1		
2		

Indice		
1		
3		
4		

UNION		
Indice		
1		
2		
1		
3		
4		

union

UNION ALL



UNION ALL Diagrama de Venn

Siguiendo la explicación de la sentencia anterior, podemos ejecutar la sentencia `UNION ALL`, permitiendo que existan duplicados en las tablas. Por lo tanto nos devuelve todos los registros de la unión de los registros de ambas tablas. Por ejemplo, si hacemos uso de las tablas del apartado anterior y modificamos ligeramente la sentencia que nos permite obtener los países para las tablas anteriores:

```
SELECT pais
FROM clientes
UNION ALL
SELECT pais
FROM proyectos;
```

país

España

Francia

Portugal

Alemania

Italia

España

Francia

Portugal

Alemania

Italia

Como podemos ver ahora se han duplicados las ciudades.

Utilizando ORDER BY y LIMIT sobre UNION / UNION ALL

Se puede utilizar las cláusulas de `ORDER BY` y `LIMIT` en conjunción con las instrucciones de tipo `UNION`, para ello debemos añadir las cláusulas de orden y límite de resultados al final de toda la sentencia.

```
SELECT columna1, ...
FROM tabla1
UNION
SELECT columna1, ...
FROM tabla2
ORDER BY columna1, ...
LIMIT numero_límite;
```

Mostramos a continuación como quedarían las tablas anteriores. Con la sentencia UNION:

```
SELECT país
FROM clientes
UNION
SELECT país
FROM proyectos
ORDER BY país;
```

ciudad

Alemania

España

Francia

Italia

Portugal

Con la sentencia UNION ALL :

```
SELECT pais
FROM clientes
UNION ALL
SELECT pais
FROM proyectos
ORDER BY pais
LIMIT 6;
```

ciudad

Alemania

Alemania

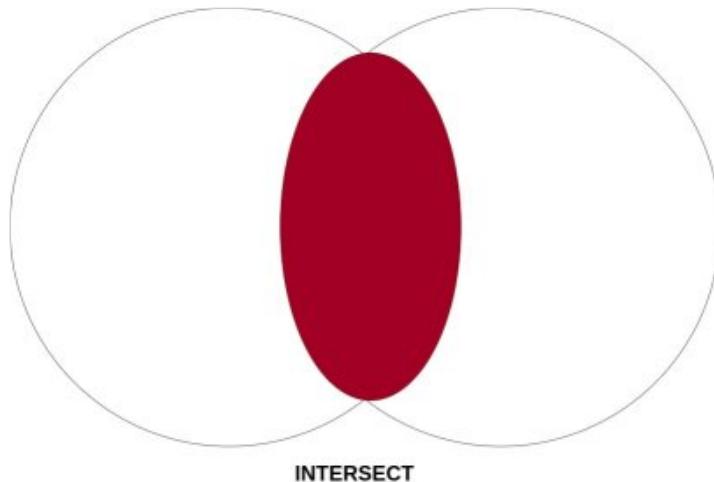
España

España

Francia

Francia

IN (INTERSECT)



UNION Diagrama de Venn

En MySQL no existe la instrucción `INTERSECT`, que suele ser bastante común en la sintaxis básica de SQL, no obstante la funcionalidad de esta se puede obtener de la misma forma haciendo uso de la sentencia `IN` en MySQL.

La instrucción `IN` (`INTERSECT`) nos devuelve las filas de la primera tabla, que son idénticas a las de la segunda tabla. A efectos prácticos, nos devuelve aquellos datos de las columnas que hayamos seleccionado y sean comunes a ambas. Así mismo no devolverá duplicados y devolverá aquellos valores coincidentes que sean nulos

La instrucción a utilizar para obtener el resultado sería la siguiente:

```
SELECT columna1
FROM tabla1
WHERE columna1 IN (
    SELECT column_name
    FROM tabla2);
```

Haciendo uso de nuevo de las tablas del apartado anterior, si ejecutamos la siguiente sentencia:

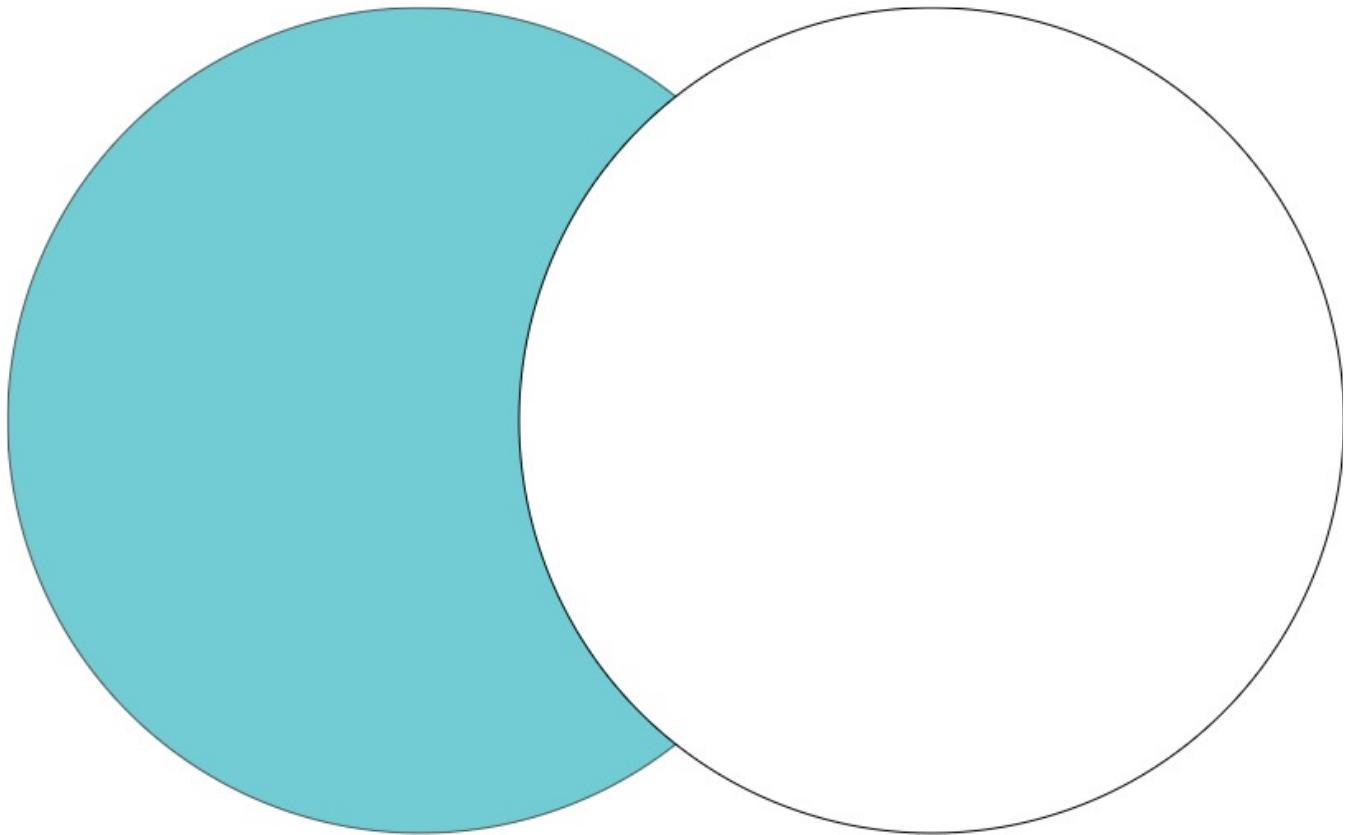
```
SELECT ciudad
FROM clientes
WHERE ciudad IN(
    SELECT ciudad
    FROM proyectos);
```

Obtendríamos como resultado lo siguiente:

ciudad
Berlin

Ya que en este caso, únicamente hay un caso en el que la ciudad aparece en ambas tablas. En el caso de que no se diera el caso de que tengamos un campo en común, nos devolvería una tabla vacía.

NOT IN (EXCEPT)



UNION Diagrama de Venn

De nuevo en MySQL no existe la sentencia `EXCEPT`, que de nuevo vuelve a ser bastante común en la sintaxis básica de SQL, por fortuna también podemos replicar la funcionalidad de esta instrucción en MySQL haciendo uso de la sentencia `NOT IN`.

La instrucción `NOT IN (EXCEPT)` nos devuelve la filas de la primera tabla, que no se pueden encontrar en la segunda tabla. A efectos prácticos, nos devuelve aquellos datos de las columnas que hayamos seleccionado y que únicamente se encuentren en la primera tabla.

```
SELECT columna1
FROM tabla1
WHERE columna1 NOT IN (
    SELECT columna1*
    FROM tabla2);
```

Haciendo uso de nuevo de las tablas anteriores.

```
SELECT ciudad
FROM clientes
WHERE ciudad NOT IN(
    SELECT ciudad
    FROM proyectos);
```

Se obtendría el siguiente resultado:

ciudad

Madrid

Paris

Lisboa

Roma

Como podemos ver la única ciudad que esta en ambas tablas es Berlin (como hemos visto también en el apartado anterior) de esta forma nos ha devuelto las ciudades de la primera tabla, salvo Berlin por encontrarse en ambas.

NOTA: Existen formas alternativas de ejecutar las sentencias de tipo EXCEPT y INTERSECT en MySQL, para ello deberemos hacer un uso elaborado de diferentes joins utilizando condiciones y filtros, en nuestro caso veremos únicamente como realizarlo con los operadores IN / NOT IN.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar una tabla ya creada llamada `customers` (clientes/as), que está en la base de datos `tienda`. Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla `customers` tiene las siguientes columnas:

- `customer_number` : el número identificativo de las cuentas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/las cuentas/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita más espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

La tabla Employees tiene las siguientes columnas:

- `employee_number` : el número identificativo de las empleadas/os. Es un número entero y sirve de clave primaria.
- `last_name` : el apellido de las empleadas. Es una cadena de texto.
- `first_name` : el nombre de las empleadas. Es una cadena de texto.
- `extension` : su extensión telefónica. Es una cadena de texto.
- `email` : el correo electrónico de la empleada. Es una cadena de texto.
- `office_code` : El código de la oficina de la empleada. Es una cadena de texto.
- `reports_to` : el número identificativo de la empleada a la que reporta (su supervisora). Es un número entero y clave foránea (relacionada con employeeNumber).
- `job_title` : el nombre del puesto de trabajo que desempeña. Es una cadena de texto.

La tabla Offices tiene las siguientes columnas:

- `office_code` : El código de la oficina. Es un número entero y sirve de clave primaria.
 - `city` : nombre de la ciudad donde se encuentra la oficina. Es una cadena de texto.
 - `phone` : Teléfono de la oficina. Es una cadena de texto.
-
- `address_line1` : La dirección de la oficina. Es una cadena de texto.
 - `address_line2` : La dirección secundaria de la oficina, por si hubiera algún carácter que no cupiera en la columna de la primera dirección. Es una cadena de texto.
 - `state` : El estado en que se encuentra dicha oficina (Para el caso de EEUU o países que agreguen otros países). Es una cadena de texto.
 - `country` : país en el cual se encuentra la oficina. Es una cadena de texto.
 - `postal_code` : El código postal de la oficina. Es una cadena de texto.
 - `territory` : Designación geográfica del territorio donde se encuentra dicha oficina. Es una cadena de texto.

EJERCICIO 1

Selecciona los apellidos que se encuentren en ambas tablas para employees y customers, con alias 'Apellidos'.

EJERCICIO 2

Selecciona los nombres con alias 'nombre' y apellidos, con alias 'Apellidos' tanto de los clientes como de los empleados de las tablas employees y customers.

EJERCICIO 3

Selecciona todos los nombres con alias 'nombre' y apellidos, con alias 'Apellidos' tanto de los clientes como de los empleados de las tablas employees y customers.

EJERCICIO 4

Queremos ver ahora que empleados tienen algún contrato asignado con alguno de los clientes existentes. Para ello selecciona employeeNumber como 'Número empleado', firstName como 'nombre Empleado' y lastName como 'Apellido Empleado'

EJERCICIO 5

Queremos ver ahora en cuantas ciudades en las cuales tenemos clientes, no también una oficina de nuestra empresa para ello: Selecciona aquellas ciudades como 'ciudad' y los nombres de las empresas como 'nombre de la empresa' de la tabla customers, sin repeticiones, que no tengan una oficina en dicha ciudad de la tabla offices.

Resumen de consultas múltiples 3

- UNION: Combina dos o más instrucciones de SELECT, nos devuelve aquellos datos comunes a ambas tablas sin duplicados.
 - Los tipos de datos de las columnas seleccionadas deben ser iguales. Eg: No podemos en una de las tablas tomar un campo como int y en otro como str.
- UNION ALL: Identico al UNION, solo que devuelve duplicados.
- Para aplicar filtros para ordenar, limitar, etc. Se han de poner al final de las dos sentencias utilizadas en el UNION.
- IN (INTERSECT): Nos selecciona aquellos registros comunes e idénticos a ambas tablas.
 - Diferencias entr IN(INTERSECT) y INNER JOIN:
 - INNER JOIN: Devuelve duplicados y puede devolver campos que contengan NULL
 - IN (INTERSECT): No devuelve duplicados y puede devolver campos con NULL.
- NOT IN (EXCEPT): Nos selecciona aquellos registros que únicamente se encuentran en una de las tablas seleccionada, considerando la primera tabla especificada como aquella sobre la cual queremos extraer los datos que no existen en la otra.

Pair programming Consultas en múltiples tablas

3

Recordad que no es necesario que hagáis todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind.

Hoy practicaremos en práctica sentencias como `UNION` , `UNION ALL` , `INTERSECT` o `EXCEPT` .

Para esta práctica te hará falta crear en algunos de los ejercicios una columna temporal.

Para ver como funciona esta creación de columnas temporales prueba el siguiente código:

```
SELECT 'Hola!' AS tipo_nombre  
FROM customers;
```

Debe devolver algo similar:

#	tipo_nombre
1	Hola!
2	Hola!
3	Hola!
4	Hola!
5	Hola!
6	Hola!
7	Hola!
8	Hola!
9	Hola!
10	Hola!

sql84

Enunciados

1. Extraer toda la información sobre las compañías que tengamos en la BBDD

Nuestros jefes nos han pedido que creamos una query que nos devuelva todos los clientes y proveedores que tenemos en la BBDD. Mostrad la ciudad a la que pertenecen, el nombre de la empresa y el nombre del contacto, además de la relación (Proveedor o Cliente). Pero importante! No debe haber duplicados en nuestra respuesta. La columna Relationship no existe y debe ser creada como columna temporal. Para ello añade el valor que le quieras dar al campo y utilizada como alias Relationship .

Nota: Deberás crear esta columna temporal en cada instrucción SELECT.

El resultado de la query debe devolver:

	City	CompanyName	ContactName	Relationship
▶	?rhus	Vaffeljernet	Palle Ibsen	Customers
	Aachen	Drachenblut Delikatessen	Sven Ottlieb	Customers
	Albuquerque	Rattlesnake Canyon Grocery	Paula Wilson	Customers
	Anchorage	Old World Delicatessen	Rene Phillips	Customers
	Ann Arbor	Grandma Kelly's Homestead	Regina Murphy	Suppliers
	Annecy	Gai pâturage	Eliane Noz	Suppliers
	Barcelona	Galería del gastrónomo	Eduardo Saavedra	Customers
	Barquisimeto	LILA-Supermercado	Carlos González	Customers
	Bend	Bigfoot Breweries	Cheryl Saylor	Suppliers
	Bergamo	Magazzini Alimentari Riuniti	Giovanni Rovelli	Customers
	Berlin	Alfreds Futterkiste	Maria Anders	Customers
	Berlin	Heli Sörbärwaren GmbH & Co...	Petra Winkler	Suppliers
	Bern	Chop-suey Chinese	Yang Wang	Customers
	Boise	Save-a-lot Markets	Jose Pavarotti	Customers
	Boston	New England Seafood Cann...	Robb Merchant	Suppliers
	Brücke	Folk och f? HB	Maria Larsson	Customers
	Brandenburg	K?niglich Essen	Philip Cramer	Customers
	Bruxelles	Maison Dewey	Catherine Dewey	Customers

sql81

2. Extraer todos los pedidos gestionados por "Nancy Davolio"

En este caso, nuestro jefe quiere saber cuantos pedidos ha gestionado "Nancy Davolio", una de nuestras empleadas. Nos pide todos los detalles tramitados por ella.

Los resultados de la query deben parecerse a la siguiente tabla:

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipPostalCode	ShipCountry
▶	10239	ERNSH	1	1996-07-17 00:00:00	1996-08-14 00:00:00	1996-07-23 00:00:00	1	140.51	Ernst Handel	Kirchgasse 6	Graz	8010	Austria	
	10270	WARTH	1	1996-08-01 00:00:00	1996-08-29 00:00:00	1996-08-02 00:00:00	1	136.54	Wartian Herku	Torkatu 38	Oulu	90110	Finland	
	10275	MAGAA	1	1996-08-07 00:00:00	1996-09-04 00:00:00	1996-08-09 00:00:00	1	26.93	Magazzini Alimentari Riuniti	Via Ludovico il Moro 22	Bergamo	24100	Italy	
	10285	QUICK	1	1996-08-20 00:00:00	1996-09-17 00:00:00	1996-08-26 00:00:00	2	76.83	QUICK-Stop	Taucherstraße 10	Cunewalde	1307	Germany	
	10292	TRADH	1	1996-08-28 00:00:00	1996-09-25 00:00:00	1996-09-02 00:00:00	2	1.35	Tradición Hipermercados	Av. Int. dos Castro, 414	Sao Paulo	SP	05634-030	Brazil
	10293	TORTU	1	1996-08-29 00:00:00	1996-09-26 00:00:00	1996-09-11 00:00:00	3	21.18	Tortuga Restaurante	Avenida Azteca 123	México D.F.	5033	Mexico	
	10304	TORTU	1	1996-09-12 00:00:00	1996-10-10 00:00:00	1996-09-17 00:00:00	2	63.79	Tortuga Restaurante	Avenida Azteca 123	México D.F.	5033	Mexico	
	10306	ROMEY	1	1996-09-16 00:00:00	1996-09-23 00:00:00	1996-09-23 00:00:00	3	7.56	Romero y tomillo	Gran Vía, 1	Madrid	28001	Spain	
	10311	DUMON	1	1996-09-20 00:00:00	1996-10-04 00:00:00	1996-09-26 00:00:00	3	24.69	Du monde entier	67, rue des Cinquante Otages	Nantes	44000	France	
	10314	RATTIC	1	1996-09-25 00:00:00	1996-10-23 00:00:00	1996-10-04 00:00:00	2	74.16	Rattlesnake Canyon Grocery	2817 Milton Dr.	Albuquerque	NM	87110	USA
	10316	RATTIC	1	1996-09-27 00:00:00	1996-10-25 00:00:00	1996-10-08 00:00:00	3	150.15	Rattlesnake Canyon Grocery	2817 Milton Dr.	Albuquerque	NM	87110	USA
	10325	KOENE	1	1996-10-09 00:00:00	1996-10-23 00:00:00	1996-10-14 00:00:00	3	64.86	Königlich Essen	Maubelstr. 90	Brandenburg		14776	Germany
	10340	BONAP	1	1996-10-29 00:00:00	1996-11-26 00:00:00	1996-11-08 00:00:00	3	166.31	Bon app'	12, rue des Bouchers	Marseille		13008	France
	10351	ERNSH	1	1996-11-11 00:00:00	1996-12-09 00:00:00	1996-11-20 00:00:00	1	162.33	Ernst Handel	Kirchgasse 6	Graz	8010	Austria	
	10357	LILAS	1	1996-11-19 00:00:00	1996-12-17 00:00:00	1996-12-02 00:00:00	3	34.88	LILA-Supermercado	Carrera 52 con Ave. Bolívar y Calle 41	Barquisimeto	Lara	3508	Venezuela
	10361	QUICK	1	1996-11-22 00:00:00	1996-12-20 00:00:00	1996-12-03 00:00:00	2	183.17	QUICK-Stop	Taucherstraße 10	Cunewalde	1307	Germany	
	10364	EASTC	1	1996-11-26 00:00:00	1997-01-07 00:00:00	1996-12-04 00:00:00	1	71.97	Eastern Connection	35 King George	London		WX3 6FW	UK
	10371	LAMAI	1	1996-12-03 00:00:00	1996-12-31 00:00:00	1996-12-24 00:00:00	1	0.45	La maison d'Asie	1 rue Alsace-Lorraine	Toulouse		31000	France
	10374	WOLZA	1	1996-12-05 00:00:00	1997-01-02 00:00:00	1996-12-09 00:00:00	3	3.94	Wolski Zajazd	ul. Filtrowa 68	Warszawa		01-012	Poland

sql82

3. Extraed todas las empresas que no han comprado en el año 1997

En este caso, nuestro jefe quiere saber cuántas empresas no han comprado en el año 1997.

Pista Para extraer el año de una fecha, podemos usar el estamento year . Más documentación sobre este método [aquí](#).

El resultado de la query será:

	CompanyName	Country
▶	Centro comercial Moctezuma	Mexico
	FISSA Fabrica Inter. Salchichas S.A.	Spain
	La corne d'abondance	France
	Paris sp?cialit?s	France
	Romero y tomillo	Spain

sql83

4. Extraed toda la información de los pedidos donde tengamos "Konbu"

Estamos muy interesados en saber como ha sido la evolución de la venta de Konbu a lo largo del tiempo. Nuestro jefe nos pide que nos muestre todos los pedidos que contengan "Konbu".

Pista En esta query tendremos que combinar conocimientos adquiridos en las lecciones anteriores como por ejemplo el `INNER JOIN`.

Los resultados nos devolverán algo como esto:

#	order_id	customer_id	employee_id	order_date	required_date	shipped_date	ship_via	freight	ship_name	ship_address	ship_city	ship_region	ship_postal_code	ship_country
1	10276	TORTU	8	1996-08-08 00:00:00	1996-08-22 00:00:00	1996-08-14 00:00:00	3	13.84	Tortuga Restaurante	Avda. Azteca 123	México D.F.	5033		Mexico
2	10291	QUEDE	6	1996-08-27 00:00:00	1996-09-24 00:00:00	1996-09-04 00:00:00	2	6.4	Que Delicia	Rua da Panificadora, 12	Rio de Janeiro	RJ	02389-673	Brazil
3	10325	KOENE	1	1996-10-09 00:00:00	1996-10-23 00:00:00	1996-10-14 00:00:00	3	64.86	Königlich Essen	Maubelestr. 90	Brandenburg		14776	Germany
4	10383	AROUT	8	1996-12-16 00:00:00	1997-01-13 00:00:00	1996-12-18 00:00:00	3	34.24	Around the Horn	Brook Farm Stratford St. Mary	Colchester	Essex	C07 6JX	UK
5	10391	DRACD	3	1996-12-23 00:00:00	1997-01-20 00:00:00	1996-12-31 00:00:00	3	5.45	Drachenblut Delikatessen	Walserweg 21	Aachen		52066	Germany
6	10394	HUNGC	1	1996-12-25 00:00:00	1997-01-22 00:00:00	1997-01-03 00:00:00	3	30.34	Hungry Coyote Import Store	City Center Plaza 516 Main St.	Elgin	OR	97827	USA
7	10420	WELLI	3	1997-01-21 00:00:00	1997-02-18 00:00:00	1997-01-27 00:00:00	1	44.12	Wellington Importadora	Rua do Mercado, 12	Resende	SP	08737-363	Brazil
8	10462	CONSH	2	1997-03-03 00:00:00	1997-03-31 00:00:00	1997-03-18 00:00:00	1	6.17	Consolidated Holdings	Berkeley Gardens 12	Brewery London		WX1 6LT	UK
9	10506	OTTIK	1	1997-04-16 00:00:00	1997-05-14 00:00:00	1997-05-13 00:00:00	2	4.99	Ottiles & seladen	Mehrheimerstr. 369	Köln		50739	Germany
10	10526	WARTH	4	1997-05-05 00:00:00	1997-06-02 00:00:00	1997-05-15 00:00:00	2	58.59	Wartian Herku	Torikatu 38	Oulu		90110	Finland

sql84

Happy coding

Consultas en múltiples tablas 4

Consultas en múltiples tablas 4: SUBQUERIES

Orden de ejecución de los elementos de la consulta

Ahora que tenemos una idea de todas las partes principales que componen una consulta SQL, podemos hablar sobre cómo encajan todas juntas en el contexto de una consulta completa.

Cada consulta comienza indicando los atributos (o datos derivados) que queremos conocer de la base de datos. Luego se filtran esos datos y se procesan si fuese necesario para que se puedan comprender de manera rápida y sencilla.

Debido a que las distintas partes de cada consulta se ejecutan secuencialmente, es importante comprender el orden de ejecución de esa secuencia para poder hacernos una impresión previa del comportamiento de la consulta. De esa manera podremos saber qué resultados son accesibles en cada momento. De lo contrario, las consultas podrían no estar comportándose como nosotras creemos que lo deberían hacer, y los resultado finales podrían no ser los esperados.

El orden de ejecución de los principales operadores de una consulta es el siguiente:

1. `FROM` y `JOINS` : se usan para seleccionar la/s tablas de las que vamos a sacar los datos.
2. `WHERE` : se utiliza para imponer condiciones a los datos de la tabla.
3. `GROUP BY` : después de imponer las condiciones, se pueden agrupar los datos según alguno de los atributos.
4. `HAVING` : después de agrupar los datos según algún atributo, se pueden poner condiciones a los resultados obtenidos.
5. `SELECT` : al terminar con todo el proceso de "eliminación" de atributos según las condiciones, se seleccionan los atributos o datos que se quieren visualizar.
6. `DISTINCT` : se indica si se quieren visualizar o no aquellos registros con valores iguales para alguno de los atributos.
7. `ORDER BY` : se ordenan los registros según los valores de alguno de los atributos seleccionados.
8. `LIMIT / OFFSET` : con `LIMIT` se limitan los resultados visualizados a cierto número de filas. También se decide con `OFFSET` si se quieren visualizar los resultados desde el primero o desde otra posición.

NOTA: No todas las consultas necesitan tener todas las partes.

Orden de escritura	Orden de ejecución
SELECT	FROM
DISTINCT	JOIN
FROM	WHERE
JOINS	GROUP BY
WHERE	HAVING
GROUP BY	SELECT
HAVING	DISTINCT
ORDER BY	ORDER BY
LIMIT - OFFSET	LIMIT - OFFSET

Tablas orden de ejecución

Ejercicio: Indica el orden de ejecución de la siguiente secuencia SQL, indicando las tablas Resultado que se van obteniendo tras cada paso.

La tabla empleadas sería la siguiente:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Por su lado, la tabla empleadas_en_proyectos contiene los siguientes pasos:

id_empleada	id_proyecto
1	1
1	2
2	1
3	2
3	3
3	5
4	2
5	3

La consulta a analizar es:

```
SELECT empleadas.id_empleada, empleadas.nombre, empleadas.apellido
FROM empleadas INNER JOIN empleadas_en_proyectos
ON empleadas.id_empleada = empleadas_en_proyectos.id_empleada
GROUP BY (empleadas.id_empleada)
HAVING COUNT(*)>1;
```

SUBCONSULTAS

Una subconsulta consiste en una instrucción `SELECT` como las que hemos visto hasta este momento situada dentro de otra instrucción `SELECT`. Las subconsultas se pueden utilizar en aquellos lugares de la consulta SQL original donde el resultado de la subconsulta (una tabla Resultado con un conjunto de valores) pueda ser usado con sentido, no en cualquier parte. Este es el caso de los operadores `HAVING` y `WHERE` donde siempre se busca que se cumpla una condición. Esta condición puede depender de un valor indicado a mano, o por el contrario de los valores resultado de otra consulta.

Hasta ahora las condiciones que hemos utilizado se han basado en valores simples como números, caracteres, o el resultado de una operación aritmética. Sin embargo, una subconsulta puede devolver registros y atributos que se utilicen también como condición. Es decir, si quisieramos imponer una condición con `WHERE` cuyos valores dependiesen de otros datos de la base de datos, podríamos ejecutar una subconsulta dentro de dicho `WHERE`.

Se suelen utilizar subconsultas en las cláusulas `WHERE` o `HAVING` cuando los datos que queremos visualizar están en una tabla, pero para seleccionar las filas de esa tabla necesitamos un dato que está en otra tabla diferente. En esta lección nos centraremos en las subconsultas realizadas sobre la cláusula `WHERE`.

No obstante también se pueden encontrar en la cláusula de `SELECT` o en la cláusula `FROM`.

IMPORTANTE: Las condiciones impuestas con WHERE y HAVING se ejecutan por cada registro de la tabla original, para comparar si cumple la condición o no. Esto significa que cuando se incluye una subconsulta dentro de una condición, dicha subconsulta se ejecutará una vez por cada fila de la consulta principal. Es importante conocer este dato ya que puede hacer que el tiempo de ejecución total sea muy elevado.

Otro beneficio del uso de subconsultas es que suelen ser bastante fáciles de interpretar por el usuario.

Si volvemos al ejemplo de las tablas empleadas y empleadas_en_proyectos podemos utilizar los resultados de una subconsulta para imponer condiciones con WHERE :

```
SELECT id_empleada, nombre
FROM empleadas
WHERE id_empleada IN (
    SELECT id_empleada
    FROM empleadas_en_proyectos
    WHERE id_proyecto=2);
```

El resultado de la consulta anterior sería el siguiente:

id_empleada	nombre
1	Ana
3	Lucía
4	Rocío

¿De esta forma que es lo que hemos hecho en la consulta anterior?

En la consulta anterior hemos buscado seleccionar el id_empleada y el nombre de aquellas empleadas pertenecientes al proyecto con id_proyecto = 2. Debido a que la información de los proyectos a los que pertenecen las empleadas se encuentra en otra tabla distinta (empleadas_en_proyectos), los hemos extraído mediante una subconsulta incluida dentro del WHERE .

La condición se busca imponer dentro del WHERE de la consulta principal es seleccionar ciertos id_empleadas, por lo que la subconsulta deberá devolver una tabla resultado que contenga id_empleadas. Para que la condición WHERE sea válida, al estar utilizando el operador IN , los valores que devuelva la subconsulta pueden estar compuestos por más de un registro. En este caso, las empleadas asociadas al id_proyecto 2 son 3 distintas (id_empleada 1, 3 y 4).

Para facilitar la compresión, vemos primero que nos devolvería el código de la subconsulta:

```
SELECT id_empleada
      FROM empleadas_en_proyectos
     WHERE id_proyecto=2;
```

El resultado de la subconsulta serían los siguientes:

id_empleada
1
3
4

La subconsulta ha seleccionado los id_empleada para aquellas que trabajan en el id_proyecto 2. Después, ese resultado se ha utilizado dentro del WHERE de la consulta principal

```
SELECT id_empleada, nombre
      FROM empleadas
     WHERE id_empleada IN (
        SELECT id_empleada
          FROM empleadas_en_proyectos
         WHERE id_proyecto=2);
```

Por lo que el resultado final de la consulta completa será:

id_empleada	nombre
1	Ana
3	Lucía
4	Rocío

Se han seleccionado el id_empleada y nombre para aquellas empleadas con id_empleada 1, 3 y 4 (ya que son aquellas que están asignadas al id_proyecto 2).

Uso de IN en subconsultas:

El operador IN se puede usar para seleccionar aquellos registros de la consulta principal para los que algún atributo de la tabla principal presenta un valor **dentro de un conjunto de valores devuelto por la subconsulta**. Es decir, la subconsulta en este caso devolvería más de un registro, y todos ellos se usarán en el WHERE o HAVING para comprobar la condición:

```

SELECT id_empleada, nombre
FROM empleadas
WHERE id_empleada IN (
    SELECT id_empleada
    FROM empleadas_en_proyectos);

```

En el ejemplo superior se han seleccionado todos los id_empleada de la tabla empleadas_en_proyectos. Despu s se utilizan esos id_empleada en el operador WHERE de la consulta principal para seleccionar las entradas de la tabla empleadas correspondientes a esos id_empleada. De esta manera estamos descartando aquellas empleadas de la tabla empleadas que no est n asignadas a ning n proyecto. El resultado ser a el siguiente:

id_empleada	nombre
1	Ana
2	Maria
3	Luc�a
4	Roc�o

NOT IN en subconsultas:

Como la intuici n nos puede indicar, se usa para invertir el funcionamiento de IN , es decir para recuperar aquellos registros de la consulta principal para los que no hay ning n registro de los devueltos por la subconsulta que contenga un valor igual.

```

SELECT id_empleada, nombre
FROM empleadas
WHERE id_empleada NOT IN (
    SELECT id_empleada
    FROM empleadas_en_proyectos);

```

En el ejemplo superior estamos realizando la misma subconsulta que para en el caso de IN : se seleccionan todos los id_empleada de la tabla empleadas_en_proyectos. Sin embargo, en este caso se utilizan dichos id_empleada para seleccionar las entradas de la tabla empleadas que NO coincidan con ninguno de ellos. De esta manera estamos seleccionando solo aquellas empleadas de la tabla empleadas que no est n asignadas a ning n proyecto, mientras que descartamos a las que s  lo est n.

En este caso el resultado de la consulta anterior nos devuelve, lo siguiente:

id_empleada	nombre
NULL	NULL

Ya que no tenemos ninguna empleada que no tenga proyecto asignado.

NOTA: La palabra `NOT` puede utilizarse delante de otras cláusulas que veremos en el curso (`ANY`, `ALL` y `EXISTS`).

Ejercicio: Selecciona el nombre, apellido y salario de las empleadas asignadas a más de un proyecto:

PISTA: Comienza pensando que consulta necesitarías para conocer el `id_empleada` de las empleadas asignadas a mas de un proyecto.

```
SELECT id_empleada  
FROM empleadas_en_proyectos  
GROUP BY id_empleada  
HAVING COUNT(id_proyecto) > 1;
```

Uso de ANY, ALL y EXISTS en subconsultas:

Se puede utilizar el operador `ANY` (o `SOME`, dependerá del SGBD usado), para recuperar registros de una consulta principal que satisfagan la condición establecida para al menos un valor de los indicados. En palabras más sencillas:

1. Se impone una condición en la consulta para uno de los atributos de la tabla.
2. Se compara el valor de dicho atributo con cada uno de los valores dentro del `ANY`.
3. Si la condición se cumple al menos para uno de los valores dentro del `ANY`, se dará por válido y se seleccionará el registro correspondiente. Es decir, la condición debe ser cierta para al menos una tupla.

Este operador es comúnmente usado con subconsultas, donde los valores indicados dentro del `ANY` son devueltos por dicha subconsulta. Veámoslo con un ejemplo:

```
SELECT id_empleada,  
       nombre,  
       salario  
  FROM empleadas AS E1  
 WHERE E1.salario > ANY (  
       SELECT salario  
         FROM empleadas AS E2  
        WHERE E2.pais = E1.pais  
     );
```

En la consulta anterior queremos seleccionar aquellas empleadas cuyo salario sea mayor que cualquiera de los salarios de las compañeras que trabajen en el mismo país. Es decir, para cada país se seleccionarán todas las empleadas menos aquella que tenga el salario más bajo ya que cuando se compare su salario con el de las demás, no será mayor que el de ninguna.

La subconsulta dentro del `ANY` devuelve una tabla Resultado que contiene los salarios de las empleadas que sean del mismo país que la empleada que se está considerando en ese momento en la consulta principal. Después se compara el salario de la empleada con cada uno de los salarios obtenidos en la subconsulta. Si se cumple que dicho salario es mayor que *el de al menos una* de esas entradas, se seleccionará esa empleada para el resultado final de la consulta.

El resultado de la consulta anterior será el siguiente:

id_empleada	nombre	salario
2	Maria	4000
3	Lucía	3000
4	Elena	5000

El uso del operador `ALL` es idéntico al del operador `ANY`, pero en este caso la condición deberá cumplirse para *todos* los registros dentro del `ALL`. Volviendo al mismo ejemplo anterior:

```
SELECT id_empleada,
       nombre,
       salario
  FROM empleadas AS E1
 WHERE E1.salario >= ALL (
           SELECT salario
  FROM empleadas AS E2
 WHERE E2.pais = E1.pais
 );
```

Ahora, el salario de cada empleada se comparará con el de todas las demás empleadas de su mismo país. Si su salario es mayor o igual al de todas las demás (es decir, si es la que más cobra) se seleccionará su registro. De esta manera, la consulta ha pasado de elegir a todas las empleadas menos la que menos cobra (con `ANY`) a seleccionar solamente la/s empleada/s que más cobran.

El resultado de la consulta anterior será el siguiente:

id_empleada	nombre	salario
4	Elena	5000
2	Rocío	1500

Ejercicio: En el `WHERE` de la anterior consulta hemos usado un `>=` y no un `>` "a secas" debido a que el salario de la empleada también se está comparando con el suyo propio. De no incluir el `=`, no se cumpliría la condición para ninguna de las empleadas. Otra solución podría consistir en añadir condiciones adicionales en el `WHERE` de la subconsulta. ¿Cómo lo harías?

El uso del operador `EXISTS` por su parte, es ligeramente diferente. En este caso, se utilizará en comparaciones contra resultados de subconsultas que sean verdadero o falso. Es decir, `EXISTS` nos indica si la subconsulta ha devuelto algún registro o por el contrario está vacía.

```

SELECT id_empleada,
       nombre,
       salario
FROM empleadas AS E1

WHERE EXISTS (
    SELECT *
    FROM empleadas AS E2
    WHERE E2.pais = E1.pais
        AND E2.id_empleada <> E1.id_empleada
);

```

En el ejemplo superior estamos intentando seleccionar los datos de aquellas empleadas que trabajan en algún país en el que haya más de una empleada (ellas mismas no cuentan debido a la condición `E2.id_empleada <> E1.id_empleada`). Para ello, en la subconsulta dentro de `EXISTS` se están extrayendo los registros de aquellas empleadas cuyo país sea el mismo que el de la empleada considerada en la consulta principal. En este caso, `EXISTS` comprobará si realmente existe algún resultado dentro de la subconsulta, y de ser así, el `WHERE` se considerará como cumplido.

De esta forma el resultado de la consulta anterior sería el siguiente:

id_empleada	nombre	salario
1	Ana	2500
2	Maria	4000
3	Lucía	3000
4	Elena	5000

Las subconsultas pueden anidarse varias veces unas dentro de otras, no solo hay un nivel de profundidad. De esta forma una subconsulta puede aparecer en la cláusula `WHERE` o `HAVING` de otra subconsulta que a su vez forme parte de otra subconsulta, etc.

Dado que (como dijimos anteriormente) cada subconsulta se ejecuta una vez por cada registro considerado en la consulta inmediatamente superior, cuantos más niveles de anidamiento presente una consulta, más tiempo y memoria va a requerir para su ejecución. Por ello, se debe pensar con cuidado si es adecuado utilizar subconsultas, especialmente cuando estas tengan varios niveles de profundidad. Adicionalmente, el hecho de introducir muchos niveles de anidamiento también hace que las consultas se puedan volver mucho más difícil de comprender por los seres humanos.

Ejercicio: Vamos a realizar un ejercicio en el que usaremos consultas y subconsultas a diferentes niveles de anidamiento y con distintos operadores de condición. Volviendo a las tablas empleadas y EmpleadasPorProyecto:

- Calcula el numero de empleadas por cada proyecto.
- Usando la consulta anterior como subconsulta, selecciona el id_proyecto del proyecto con el mayor numero de empleadas.
- Por último, usa todas las consultas anteriores para seleccionar el id_empleada, nombre y apellido de las empleadas asignadas al proyecto con mayor numero de empleadas.

ENUNCIADO EJERCICIOS

En este ejercicio vamos a usar unas tablas ya creadas llamadas `customers` (clientes/as) y `employees` (empleados/as), que está en la base de datos `tienda`. Antes de escribir vuestra query cargamos la base de datos `tienda` escribiendo `USE tienda;` en la primera línea. Como tenemos múltiples bases de datos en nuestro MySQL Workbench, así le decimos en qué base de data debe buscar la tabla que especificamos en nuestro código.

La tabla customers tiene las siguientes columnas:

- `customer_number` : el número identificativo de las clientas/es. Es un número entero y sirve de clave primaria.
- `customer_name` : el nombre de las empresas en las que trabajan las/los clientas/es. Es una cadena de texto.
- `contact_last_name` : El apellido de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `contact_first_name` : El nombre de la persona de contacto en la empresa cliente. Es una cadena de texto.
- `phone` : El teléfono de la persona de contacto en la empresa cliente. Es una cadena de texto (ya que hay espacios).
- `address_line1` : La dirección (calle, número, etc.) de la empresa cliente. Es una cadena de texto.
- `address_line2` : La dirección de la empresa cliente (si se necesita mas espacio). Es una cadena de texto. Muchas veces está vacía.
- `city` : La ciudad de la empresa cliente. Es una cadena de texto.
- `state` : El estado en el que se encuentra la empresa cliente. Válido para los Estados Unidos. Es una cadena de texto.
- `postal_code` : El código postal. Es una cadena de texto (ya que puede haber espacios).
- `country` : El país de la empresa cliente. Es una cadena de texto.
- `sales_rep_employee_number` : El número identificador de la empleada o empleado que lleva a esa empresa cliente. Es un número entero.
- `credit_limit` : El límite de crédito que tiene la empresa cliente. Es un número decimal.

La tabla Employees tiene las siguientes columnas:

- `employee_number` : el número identificativo de las empleadas/os. Es un número entero y sirve de clave primaria.
- `last_name` : el apellido de las empleadas. Es una cadena de texto.
- `first_name` : el nombre de las empleadas. Es una cadena de texto.
- `extension` : su extensión telefónica. Es una cadena de texto.
- `email` : el correo electrónico de la empleada. Es una cadena de texto.
- `office_code` : El código de la oficina de la empleada. Es una cadena de texto.
- `reports_to` : el número identificativo de la empleada a la que reporta (su supervisora). Es un número entero y clave foránea (relacionada con employeeNumber).
- `job_title` : el nombre del puesto de trabajo que desempeña. Es una cadena de texto.

EJERCICIO 1

- Calcula el numero de clientes por cada ciudad.

EJERCICIO 2

- Usando la consulta anterior como subconsulta, selecciona la ciudad con el mayor numero de clientes.

EJERCICIO 3

- Por último, usa todas las consultas anteriores para seleccionar el customerNumber, nombre y apellido de las clientas asignadas a la ciudad con mayor numero de clientas.

Resumen consultas múltiples 4

- Subconsultas: Son consultas que se realizan dentro de otras consultas.
 - Se pueden realizar subconsultas en las siguientes cláusulas:
 - SELECT
 - FROM
 - WHERE- HAVING: Se ejecutará la subconsulta cada vez que cambiemos de registro en la consulta principal y esto puede hacer que los tiempos de ejecución sean más largos.
- Operadores adicionales en subconsultas:
 - IN: Selecciona los registros de la consulta principal, en los cuales presenta un valor dentro de los valores devueltos por la subconsulta.
 - NOT IN: Nos hace la operación inversa del IN, nos devuelve en la consulta principal los registros no incluidos en la subconsulta.
 - ANY: Devuelve los registros que cumplen las condiciones establecidas para al menos uno de sus registros.
 - ALL: Implica que se debe cumplir la condición para todos los valores de la subconsulta.
 - EXIST: Nos indica si la consulta ha devuelto algún resultado.

Pair programming Consultas en múltiples tablas

4

Recordad que no es necesario que hagas todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

Es el turno de las *subqueries*. En este ejercicios os planteamos una serie de queries que nos permitirán conocer información de la base de datos, que tendréis que solucionar usando *subqueries*.

Ejercicios

1. Extraed información de los productos "Beverages"

En este caso nuestro jefe nos pide que le devolvamos toda la información necesaria para identificar un tipo de producto. En concreto, tienen especial interés por los productos con categoría "Beverages".

Devuelve el ID del producto, el nombre del producto y su ID de categoría.

La query debería resultar en una tabla como esta:

	ProductID	ProductName	CategoryID
▶	1	Chai	1
	2	Chang	1
	24	Guaraná Fantástica	1
	34	Sasquatch Ale	1
	35	Steeleye Stout	1
	38	Côte de Blaye	1
	39	Chartreuse verte	1
	43	Ipoh Coffee	1
	67	Laughing Lumberjack Lager	1
	70	Outback Lager	1
	75	Rhönbräu Klosterbier	1
	76	Lakkaliköri	1
*	HULL	HULL	HULL

sql91

2. Extraed la lista de países donde viven los clientes, pero no hay ningún proveedor ubicado en ese país

Suponemos que si se trata de ofrecer un mejor tiempo de entrega a los clientes, entonces podría dirigirse a estos países para buscar proveedores adicionales.

Los resultados de esta query son:

	country
▶	Mexico
	Argentina
	Switzerland
	Austria
	Portugal
	Venezuela
	Ireland
	Belgium
	Poland

sql92

3. Extraer los clientes que compraron mas de 20 artículos "Grandma's Boysenberry Spread"

Extraed el OrderId y el nombre del cliente que pidieron más de 20 artículos del producto "Grandma's Boysenberry Spread" (ProductID 6) en un solo pedido.

Resultado de nuestra query deberíamos tener una tabla como esta:

	OrderID	CustomerID
▶	10871	BONAP
	10734	GOURL
	10309	HUNGO
	10618	MEREP
	10989	QUEDE
*	HULL	HULL

sql93

4. Extraed los 10 productos más caros

Nos siguen pidiendo más queries correlacionadas. En este caso queremos saber cuáles son los 10 productos más caros.

Los resultados esperados de esta query son:

	Ten_Most_Expensive_Products	UnitPrice
▶	Côte de Blaye	263.5
	Thüringer Rostbratwurst	123.79
	Mishi Kobe Niku	97

Sir Rodney's Marmalade	81
Carnarvon Tigers	62.5
Radette Courdavault	55
Manjimup Dried Apples	53
Tarte au sucre	49.3
Ipoh Coffee	46
Rössle Sauerkraut	45.6

sql94

BONUS:

1. Qué producto es más popular

Extraed cuál es el producto que más ha sido comprado y la cantidad que se compró.

El resultado de esta query es:

	ProductName	MAX(SumQuantity)
▶	Queso Cabrales	1577

sql95

Happy coding

Consultas en múltiples tablas 5

Consultas en múltiples tablas 5

Llegados a este punto ya sabemos y entendemos lo que son las subconsultas y su utilidad, de las lecciones anteriores. De esta forma vamos a explicar un par de formas adicionales de realizar subconsultas, que resultan extremadamente útiles. Procedemos pues a explicar las llamadas consultas correlacionadas.

Correlated subqueries

Las correlated subqueries (subconsultas correlacionadas), son un tipo de subconsultas donde se hace uso de los resultados de la consulta externa para la obtención de los resultados finales. Dicho de otra forma, las subconsultas correlacionadas no se pueden evaluar de forma independiente, ya que necesita de los valores de la consulta a la que pertenecen, para evaluar las condiciones impuestas para la evaluación de la subconsulta.

IMPORTANTE: Cuando se ejecuta una consulta que contiene una subconsulta correlacionada, la subconsulta se ejecuta por cada fila de la consulta principal. Es importante conocer este dato ya que puede hacer que el tiempo de ejecución total sea muy elevado.

A consecuencia de esto, la subconsulta se ejecutará varias veces (de forma repetitiva) para cada fila seleccionada en la consulta externa. Lo cual nos permitirá realizar cálculos en función de los valores de la consulta externa. Cuando estamos tratando con tablas relativamente pequeñas el tiempo de cómputo de las consultas correlacionadas no suele ser demasiado elevado. No obstante hay que tener cuidado de ejecutar subconsultas correlacionadas sobre bases de datos y tablas muy grandes, ya que como se ha de ejecutar la subconsulta cada vez que cambiamos de fila en la consulta externa, esto puede llevar a que los tiempos de cálculo se vuelvan extremadamente grandes.

Ejemplo: Imagina que estamos trabajando con una tabla que contiene un millón de registros y queremos filtrar los datos en función de los valores de la media de los valores de esa misma columna. Esto implica que para el millón de filas de la consulta, se va a ejecutar un millón de veces la subconsulta que obtiene la media de todos los valores que existen en esa misma columna. Esto puede llevar a que en lugar de tardar unos milisegundos en obtener los resultados, pueda dispararse a minutos o horas (siempre considerando el tamaño de la base de datos con la que estamos trabajando). En este tipo de casos, se puede obtener los mismo resultados haciendo uso de tablas intermedias para guardar los resultados de la subconsulta y luego contrastarlo con los valores de la consulta principal. Por lo tanto deberíamos utilizar varios pasos para obtener los resultados. No es tan compacto o elegante como realizarlo todo en una misma consulta, pero los tiempos de procesado son infinitamente menores en el caso de utilizar una subconsulta correlacionada en este tipo de circunstancias.

Vamos a ver ahora cual es la forma general de escribir una subconsulta correlacionada:

```

SELECT * (o columnas)
FROM tabla1 AS t1
WHERE columna o condición (
    SELECT columna o operacion (columna)

    FROM tabla2 AS t2
    WHERE t1.id = t2.id)

```

Vamos a ilustrar como realizar una subconsulta haciendo uso de las tablas que hemos estado utilizando hasta ahora en el resto de lecciones. Se hará uso de la tabla Empleada, que ahora contiene los siguientes campos:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia
6	2000	Inés	Romero	ines@adalab.es	619739261	Sevilla	España
7	2200	Alba	Fernández	alba@adalab.es	null	null	España
8	1800	Julia	Aguilar	julia@adalab.es	614339261	Zaragoza	España
9	2000	Irene	Montenegro	irene@adalab.es	659745615	Cataluña	España
10	3000	Laura	Navarro	laura@adalab.es	null	null	Italia

De esta forma si deseamos conocer el ID de las empleadas, su nombre, apellidos y salario, así como las empleadas cuyo salarios sea superior o igual al salario medio de todas las empleadas de un mismo país. Podríamos utilizar la siguiente subconsulta correlacionada:

```

SELECT id_empleada,
       nombre,
       apellido,
       e1.salario,
       e1.pais
  FROM empleadas AS e1
 WHERE e1.salario >= (
   SELECT AVG(e2.salario)
     FROM empleadas AS e2
    WHERE e1.pais = e2.pais)
 ORDER BY salario;

```

Donde el resultado obtenido sería el siguiente:

id_empleada	nombre	apellido	salario	pais
5	Rocío	García	1500	Francia
1	Ana	González	2500	España
10	Laura	Navarro	3000	Italia
2	Maria	López	4000	España
4	Elena	Bueno	5000	España

Como podemos observar nos ha devuelto la tabla con el id_empleada, nombre, apellido y salario, para aquellos casos en los que el salario de la empleada es mayor o igual que el salario medio de todas las empleadas, siendo este salario medio de 2700 euros.

LIKE y NOT LIKE

LIKE

Ahora que hemos llegado a este punto donde ya conocemos un montón de formas de realizar consultas sobre las bases de datos, vamos a introducir una última instrucción que habíamos evitado explicar hasta este momento y que nos permitirá poder filtrar las consultas que contengan alguna columna de tipo texto. La letra o string que deseemos buscar siempre deberá ir entre comillas simples ' '.

De esta forma el requisito mínimo será que necesitaremos realizar la consulta sobre al menos una columna con campo textual.

La sintaxis general para este tipo de consultas es la siguiente:

```

SELECT columna1, columna2, ...
FROM tabla
WHERE columna LIKE patron;

```

Donde siempre podemos incluir mas de una condición para el WHERE .

Para poder utilizar patrones en las consultas de tipo LIKE , es necesario antes conocer que existen formas diferentes de escribir patrones en este tipo de instrucciones, para ello siempre será necesario utilizar las llamados comodines (*wildcards*), que son los que permite a MySQL filtrar utilizando cadenas de texto.

Existen dos tipos de comodines:

- % Sirve para que las coincidencias sean de un string de cero o más caracteres.
- _ Este realiza las coincidencias para cualquier carácter individual.

A la hora de realizar este tipo de consultas, los resultados que obtendremos serán dependiente de las posiciones de aquellas letras o aquel string que pongamos entre las comillas simples.

En la siguiente tabla ilustramos como utilizar los comodines para encontrar patrones en las filas.

Patrón	Explicación
A%	El valor a filtrar comienza con la 'A'
%a	El valor a filtrar termina en 'a'
%a%	El valor a filtrar contiene la 'a' en cualquier posición
_a%	El valor a filtrar contiene una a en la segunda posición
e_%	El valor a filtrar empieza por e y tiene al menos dos caracteres
a__	El valor a filtrar empieza por a y tiene 3 caracteres
A%a	El valor a filtrar comienza por 'a' y termina en 'a'

Vamos a ilustrarlo con las tablas ejemplo que ya hemos utilizado, haremos uso de la tabla de empleadas.

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia

Haciendo uso de los diferentes patrones, vamos a ver como quedarían las siguientes consultas:

```
SELECT *
FROM empleadas
WHERE nombre LIKE 'A%';
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España

Nos toma únicamente aquella empleada que empieza por 'A'.

Veamos ahora otra consulta

```
SELECT *
FROM empleadas
WHERE nombre LIKE 'e_%';
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España

En este caso nos devuelve aquella empleada que empieza por 'e' y tiene una longitud de al menos 2 caracteres.

Finalmente vamos a ver, como quedaría la siguiente consulta.

```
SELECT *
FROM empleadas
WHERE nombre LIKE '%l%';
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
3	3000	Lucía	Ramos	lucia@adalab.es	674459123	Valencia	España
4	5000	Elena	Bueno	elena@adalab.es	628546577	Bilbao	España

Como podemos ver nos ha devuelto los casos de aquellas empleadas que contiene una 'l' en alguna posición del nombre.

Ejercicio: Prueba el resto de patrones de la tabla de comodines, para ver los resultados obtenidos. (Puedes copiar el patron directamente de la lista de ejemplos, ya que funcionan todos con la tabla de empleadas).

NOT LIKE

Ahora también, podemos utilizar la instrucción `NOT LIKE` para hacer que nos filtre los datos excluyendo el valor entre comillas.

```
SELECT columna1, columna2, ...
FROM tabla
WHERE columna NOT LIKE patron;
```

Lo ilustramos tomando una de las consultas mostradas anteriormente para ver que ocurriría:

```
SELECT *
FROM empleadas
WHERE nombre NOT LIKE '%l%';
```

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@adalab.es	654785214	Madrid	España
2	4000	Maria	López	maria@adalab.es	689656322	Barcelona	España
5	1500	Rocío	García	rocio@adalab.es	616365624	Paris	Francia

En este caso nos selecciona todas aquellas empleadas que no contienen una 'l' en el nombre.

Finalmente vamos a explicar un concepto relevante, algunas veces los patrones que estamos buscando pueden contener caracteres que usualmente están reservados para las sentencias SQL.

Vamos a incluir algunos de ellos, los que nos podrían afectar en mayor medida antes o después:

Secuencia de escape	Explicación
\0	Carácter nulo de ASCII (X'00')
\'	Carácter de comillas simple ('')
\"	Carácter de comillas dobles ("")
	Carácter de nueva línea
\\	Carácter de barra invertida (\) character
\%	El símbolo de porcentaje
_	El Carácter de barra baja

En estos casos hay que utilizar lo que se llama cláusula de escape, representada por \ .

Ilustramos un par de ejemplos en la parte de los ejercicios, donde únicamente tendrás que copiar la sentencia y ver los resultados.

REGEX

También podemos filtrar columnas haciendo uso de Regular Expressions (Regex), para filtrar los datos. Hacemos hincapié que este caso no se le da importancia a conocer al dedillo el funcionamiento de las regex, ya que se pueden volver extremadamente complejas. Ninguna persona en su sano juicio sabe de memoria como utilizar regex con detalle, siempre es necesario tener chuletas de el uso de las regex. Más adelante dedicaremos una clase entera a Regex en Python.

[Chuleta Regex](#)

La sintaxis básica para el uso de las regex en SQL sería la siguiente:

```
SELECT columna1, columna2, ...
FROM tabla
WHERE columna REGEXP patron_regex;
```

Por ejemplo si quisiéramos, de nuevo, volver a obtener los nombres de las empleadas que comienzan por 'A', podríamos hacer uso de la siguiente instrucción:

```
SELECT *
FROM empleadas
WHERE nombre REGEXP '^A';
```

Que nos devolvería lo siguiente:

id_empleada	salario	nombre	apellido	email	telefono	ciudad	pais
1	2500	Ana	González	ana@ad.alab.es	654785214	Madrid	España

EJERCICIO 1: Consultas correlacionadas

Selecciona de la tabla products el productCode, productName, quantityInStock, productLine, MSRP, buyPrice, para aquellos casos en los que la diferencia entre el MSRP y el precio de compra sea mayor o igual a la media de la diferencia de precios entre MSRP y precio de compra con alias 'AvgProfit', siempre que el 'AvgProfit' sea superior a 50 euros.

EJERCICIO 2:

Encuentra los campos nombre del cliente y ciudad, de aquellas ciudades de la tabla de customers que terminen en 'on'.

EJERCICIO 3:

Encuentra los campos nombre del cliente, ciudad de aquellas ciudades de la tabla de customers que terminen en 'on' y que únicamente sean de longitud 4.

EJERCICIO 4:

Encuentra el nombre del cliente, primera dirección y ciudad de aquellas ciudades que contengan el número 3 en su dirección postal (o lo que es lo mismo, su primera dirección).

EJERCICIO 5:

Encuentra el nombre del cliente, primera dirección y ciudad de aquellas ciudades que contengan el número 3 en su dirección postal y la ciudad no empiece por T.

EJERCICIO 6:

Selecciona, haciendo uso de expresiones regulares, los campos nombre del cliente, primera dirección y ciudad. Unicamente en el caso que la dirección postal, posea algún número en dicho campo.

EJERCICIO 7:

Investiga que ocurre al ejecutar la siguiente sentencia sobre la tabla de products.

```
SELECT * FROM products  
WHERE productDescription LIKE '%!%';
```

La sentencia correcta debería utilizar el carácter de escape \ , ya que las comillas simples las interpretaría como la apertura de algún texto.

```
SELECT * FROM products  
WHERE productDescription LIKE '%\'%';
```

Resumen consultas multiples 5

- Consultas correlacionadas: Son un tipo de subconsulta, en las cuales los resultados son dependientes de los valores de la consulta principal.
 - Hay que tener precaución a la hora de aplicar consultas correlacionadas sobre tablas muy grandes, ya que la subconsulta se ejecutará para cada registro de la tabla principal.
 - Para ser considerada subconsulta correlacionada, deberemos tener un campo WHERE donde se hace referencia a otro campo de la consulta principal. Ej: WHERE tabla1.id = tabla2.id.
- LIKE / NOT LIKE: Se utiliza para poder filtrar columnas que tengan datos de tipo texto.
 - El patrón (o valor) que pongamos para filtrar por LIKE deberá ir entre comillas simples " o comillas dobles "" .
 - Con '%' : Se utiliza para coincidencias de cero o más caracteres.
 - Con '_' para coincidencias de carectires individuales.
 - Las busquedas serán diferentes en función de las posiciones de '%' y '_'
- REGEX: Se utiliza para filtrar por valores de tipo texto, es mucho más versatil que LIKE, pero a la par más complejo.

Pair programming Consultas en múltiples tablas

5

Recordad que no es necesario que hagas todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

En esta lección de pair programming vamos a continuar trabajando sobre la base de datos Northwind. En este caso trabajaremos con *queries* correlacionadas y con estamenteos como el `like` y `not like`. Para solucionar algunos de los ejercicios tendremos que aplicar conocimientos apredidos previamente como los `join`.

Es el momento de poner práctica los conceptos aprendidos en la lección de *queries* múltiples V.

Ejercicios

- Extraed los pedidos con el máximo "order_date" para cada empleado.

Nuestro jefe quiere saber la fecha de los pedidos más recientes que ha gestionado cada empleado.

Para eso nos pide que lo hagamos con una query correlacionada.

Los resultados de esta *query* serán:

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
▶	11043	SPECD	5	1998-04-22 00:00:00	1998-05-20 00:00:00
	11045	BOTTM	6	1998-04-23 00:00:00	1998-05-21 00:00:00
	11058	BLAUS	9	1998-04-29 00:00:00	1998-05-27 00:00:00
	11063	HUNGO	3	1998-04-30 00:00:00	1998-05-28 00:00:00
	11070	LEHMS	2	1998-05-05 00:00:00	1998-06-02 00:00:00
	11073	PERIC	2	1998-05-05 00:00:00	1998-06-02 00:00:00
	11074	SIMOB	7	1998-05-06 00:00:00	1998-06-03 00:00:00
	11075	RICSU	8	1998-05-06 00:00:00	1998-06-03 00:00:00
	11076	BONAP	4	1998-05-06 00:00:00	1998-06-03 00:00:00
*	11077	RATTC	1	1998-05-06 00:00:00	1998-06-03 00:00:00
	NULL	NULL	NULL	NULL	NULL

sql101

- Extraed el precio unitario máximo (*unit_price*) de cada producto vendido.

Supongamos que ahora nuestro jefe quiere un informe de los productos vendidos y su precio unitario.

De nuevo lo tendrás que hacer con *queries* correlacionadas.

Los resultados deberán ser:

	ProductID	Max_unit_price_sold
▶	1	18
	2	19
	3	10
	4	22
	5	21.35
	6	25
	7	30
	8	40
	9	97
	10	31
	11	21
	12	38
	13	6
	14	23.25
	15	15.5
	16	17.45
	17	39
	18	62.5
	19	9.2
	20	81
	21	10

sql102

- Ciudades que empiezan con "A" o "B".

Por un extraño motivo, nuestro jefe quiere que le devolvamos una tabla con aquellas compañías que están afincadas en ciudades que empiezan por "A" o "B". Necesita que le devolvamos la ciudad, el nombre de la compañía y el nombre de contacto.

Los resultados deberán ser:

	City	CompanyName	ContactName
▶	Aachen	Drachenblut Delikatessen	Sven Ottlieb
	Albuquerque	Rattlesnake Canyon Grocery	Paula Wilson
	Anchorage	Old World Delicatessen	Rene Phillips
	Barcelona	Galer?a del gastr?nomo	Eduardo Saavedra
	Barquisimeto	LILA-Supermercado	Carlos Gonz?lez
	Bergamo	Magazzini Alimentari Riuniti	Giovanni Rovelli
	Berlin	Alfreds Futterkiste	Maria Anders
	Bern	Chop-suey Chinese	Yang Wang
	Boise	Save-a-lot Markets	Jose Pavarotti
	Br?cke	Folk och f? HB	Maria Larsson

	Brandenburg	K?niglich Essen	Philip Cramer
	Bruxelles	Maison Dewey	Catherine Dewey
	Buenos Aires	Cactus Comidas para llevar	Patricia Simpson
	Buenos Aires	Oc?ano Atl?ntico Ltda.	Yvonne Moncada
	Buenos Aires	Rancho grande	Sergio Guti?rrez
	Butte	The Cracker Box	Liu Wong

sql103

4. Número de pedidos que han hecho en las ciudades que empiezan con L.

En este caso, nuestro objetivo es devolver los mismos campos que en la query anterior el número de total de pedidos que han hecho todas las ciudades que empiezan por "L".

Deberéis tener una tabla como la siguiente:

	ciudad	empresa	persona_contacto	numero_pedidos
▶	London	Around the Horn	Thomas Hardy	13
	Lule?	Berglunds snabbk?p	Christina Berglund	18
	London	B's Beverages	Victoria Ashworth	10
	London	Consolidated Holdings	Elizabeth Brown	3
	London	Eastern Connection	Ann Devon	8
	Lille	Folies gourmandes	Martine Ranc?	5
	Lisboa	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	8
	Leipzig	Morgenstern Gesundkost	Alexander Feuer	5
	London	North/South	Simon Crowther	3
	Lisboa	Princesa Isabel Vinhos	Isabel de Castro	5
	London	Seven Seas Imports	Hari Kumar	9
	Lander	Split Rail Beer & Ale	Art Braunschweiger	9
	Lyon	Victuailles en stock	Mary Saveley	10

sql104

5. Todos los clientes cuyo "contact_title" no incluya "Sales".

Nuestro objetivo es extraer los clientes que no tienen el contacto "Sales" en su "contact_title". Extraer el nombre de contacto, su posición (*contact_title*) y el nombre de la compañía.

Los resultados son:

#	contact_name	contact_title	company_name
1	Ana Trujillo	Owner	Ana Trujillo Emparedados y helados
2	Antonio Moreno	Owner	Antonio Moreno Taquer?a
3	Christina Berglund	Order Administrator	Berglunds snabbk?p
4	Fr?d?rique Citeaux	Marketing Manager	Blondesddsl p?re et fils
5	Mart?n Sommer	Owner	B?lido Comidas preparadas
6	Laurence Lebihan	Owner	Bon app'
7	Elizabeth Lincoln	Accounting Manager	Bottom-Dollar Markets
8	Francisco Chang	Marketing Manager	Centro comercial Moctezuma
9	Yang Wang	Owner	Chop-suey Chinese
10	Sven Ottlieb	Order Administrator	Drachenblut Delikatessen
11	Janina Labrune	Owner	Du monde entier

sql105

6. Todos los clientes que no tengan una "A" en segunda posición en su nombre.

Devolved únicamente el nombre de contacto.

Los resultados son:

ContactName
▶ Ana Trujillo
Antonio Moreno
Thomas Hardy
Christina Berglund
Fr?d?rique Citeaux
Elizabeth Lincoln
Victoria Ashworth
Francisco Chang
Pedro ?faro

Elizabeth Brown
Sven Ottlieb
Ann Devon
Roland Mendel
Aria Cruz
Diego Roel
Peter Franken
Lino Rodriguez
Eduardo Saavedra
Isabel Dodro Freire

sql106

Happy coding

Simplificando consultas con CTEs

Simplificando consultas: Expresiones comunes de tabla (CTEs)

¿En qué consiste una expresión común de tabla?

Una expresión común de tabla (CTE por sus siglas en inglés, Common Table Expression) consiste en un conjunto de resultados temporales a los que se les asigna un nombre para poder referenciarlos en otra instrucción SQL a posteriori. De esta manera se simplifica mucho el código SQL ya que permite reutilizar trozos del mismo que sean recurrentes.

En esta sección vamos a ver cómo podemos sacar provecho de las CTE.

¿Por qué usar CTEs?

En muchas de los casos a lo largo de la lección, pensaréis que la misma funcionalidad que obtenemos con CTEs puede conseguirse usando una o más subconsultas. Sin embargo, es importante tener en cuenta que el uso de CTEs conlleva varios beneficios respecto a la opción basada en subqueries:

- Las CTEs ayudan a organizar mejor las consultas largas. Al tener que definir solo una vez el código dentro de la CTE, conseguimos reducir el tamaño de la consulta.
- Las CTEs también consiguen que las consultas sean más legibles, gracias a su modularidad y al uso de alias.
- Las CTEs reflejan mejor la lógica humana. Es decir, con ellas se define primero el resultado temporal que vamos a usar. Después vamos haciendo referencia al mismo a lo largo del código. Si usásemos varias subqueries, los resultados temporales se van definiendo en medio del código.
- Existe un tipo de CTEs llamadas CTEs recursivas que tienen permitido referenciarse a sí mismas. Este tipo de CTEs pueden resolver problemas que de otra manera no podrían acometerse. Son especialmente útiles cuando trabajamos con datos jerárquicos en la base de datos (por ejemplo si tenemos tablas con empleados asignados a supervisores que son empleados también).

Sintaxis de las CTE

A continuación tenemos un ejemplo de CTE:

```
WITH nombre_CTE [(nombres_columnas)]
AS (consulta_CTE);
```

Una CTE se debe empezar con el operador `WITH` tras el cual se indica el nombre que le queremos asignar a la expresión. Este nombre es el que se debe utilizar posteriormente para llamar a este CTE desde otras sentencias SQL. Después del nombre de la expresión, se pueden añadir de manera opcional nombres de las columnas resultado de la consulta CTE. De esta manera esos alias también podrán usarse más tarde en las consultas. Finalmente, después del operador `AS` se incluye el código de la consulta SQL que queremos almacenar como CTE.

Una cláusula `WITH` puede utilizarse en los siguientes contextos:

Antes de una consulta `SELECT`, `UPDATE` o `DELETE`

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

Por ejemplo, para utilizar una CTE que hemos definido en una sentencia `SELECT` posterior lo realizaremos de la siguiente manera:

```
SELECT [(nombres_columnas)]
FROM nombre_CTE;
```

Se indicará el nombre de la CTE en el lugar en el que normalmente va el nombre de la tabla sobre la que queremos realizar la consulta. Los alias dados a las columnas de la CTE también pueden usarse en el `SELECT` para seleccionar los atributos que queremos en el resultado.

Ejemplo: Define una CTE que calcule el salario medio de las empleadas de cada país. Después usa dicha CTE para mostrar los datos de cada empleada junto con el salario medio de su país.

La tabla empleadas sería la siguiente:

id_empleada	salario	nombre	apellido	pais
1	2500	Ana	González	España
2	4000	Maria	López	España
3	3000	Lucía	Ramos	España
4	5000	Elena	Bueno	España
5	1500	Rocío	García	Francia

Por su lado, la tabla empleadas_en_proyectos contiene los siguientes pasos:

<u>id_empleada</u>	<u>id_proyecto</u>
1	1
1	2
2	1
3	2
3	3
3	5
4	2
5	3

Para calcular el salario medio por país necesitaríamos una consulta SQL como la siguiente:

```
SELECT pais, AVG(salario) as salario_medio_pais
FROM empleadas
GROUP BY pais;
```

La consulta anterior agrupa los registros de empleada por país y calcula el salario medio para cada uno de los grupos. Después podríamos definir una CTE con el código SQL anterior y usarlo para la consulta final:

```
WITH salario_avg
AS (
    SELECT pais,
    AVG(salario) AS salario_medio_pais
    FROM empleadas
    GROUP BY pais
)
SELECT e.id_empleada,
    e.nombre,
    e.apellido,
    e.pais,
    e.salario,
    s.salario_medio_pais
FROM empleadas AS e
JOIN salario_avg AS s ON e.pais = s.pais;
```

La consulta anterior nos devuelve el siguiente resultado:

<code>id_empleada</code>	<code>nombre</code>	<code>apellido</code>	<code>pais</code>	<code>salario</code>	<code>salario_mec_o_pais</code>
1	Ana	González	España	2500	3625
2	Maria	López	España	4000	3625
3	Lucía	Ramos	España	3000	3625
4	Elena	Bueno	España	5000	3625
5	Rocío	García	Francia	1500	1500

Dentro del `WITH` hemos insertado el código correspondiente al cálculo del salario medio por país, asignándole el alias `salario_avg` a dicha CTE. Despues, hemos definido otra consulta SQL que hace uso de dicha CTE para extraer los datos de cada empleada (`ID`, `nombre`, `apellido`, `salario`, `país`) y los muestra junto al salario medio por pais. En este caso se ha usado la CTE como otra tabla con la que hacer `INNER JOIN` con la tabla `empleadas`. Usando el operador `ON` (como vimos en la lección de joins), los registros de ambas tablas se han emparejado usando las columnas País.

Al comienzo de subconsultas dentro de otra consulta `SELECT`

```
SELECT ...
WHERE id IN (
    WITH ...
    SELECT ...
) ...
```

```
SELECT ...
FROM (
    WITH ...
    SELECT ...
) AS dt ...
```

Como vimos en anteriores lecciones del temario sobre SQL, dentro de sentencias `SELECT` puede, por ejemplo, reemplazarse la tabla sobre la que normalmente se realizaría la consulta por la tabla-resultado de otra consulta `SELECT` (esta sería la mencionada subconsulta). Si usamos CTEs con `WITH`, podría reemplazarse dicha subconsulta por una CTE.

Ejemplo: Indica el numero de empleadas de cada país que cobran un sueldo por encima de la media de su pais.

Lo primero que vamos a necesitar es el código SQL del anterior ejercicio en el que extraímos los datos de cada empleada junto al salario medio de su país usando CTEs:

```

WITH salario_avg
AS (
    SELECT pais,
           AVG(salario) AS salario_avg_pais
    FROM empleadas
   GROUP BY pais
)
SELECT e.id_empleada,
       e.nombre,
       e.apellido,
       e.pais,
       e.salario AS salario,
       s.salario_avg_pais AS salario_medio
  FROM empleadas AS e
 JOIN salario_avg AS s ON s.pais = e.pais);

```

La consulta anterior nos devuelve el siguiente resultado, como en el ejemplo anterior:

id_empleada	nombre	apellido	pais	salario	salario_medio
1	Ana	González	España	2500	3625
2	Maria	López	España	4000	3625
3	Lucía	Ramos	España	3000	3625
4	Elena	Bueno	España	5000	3625
5	Rocío	García	Francia	1500	1500

Este trozo de código SQL con una CTE puede usarse dentro del `FROM` de una consulta superior. En este caso nosotras lo usaremos para comprobar por cada empleada si su salario es mayor que el salario medio de su país. Luego agruparemos los resultados por país y haremos un recuento final:

```

SELECT COUNT(id_empleada) AS NumeroEmpleadas, pais
  FROM (
    WITH salario_avg AS (
      SELECT pais, AVG(salario) AS salario_avg_pais
        FROM empleadas
       GROUP BY pais)
    SELECT e.id_empleada, e.nombre, e.apellido, e.pais, e.salario AS salario, s.salario_avg_pais
      FROM empleadas AS e
     JOIN salario_avg AS s
       ON s.pais = e.pais) AS empleadas_salarios
 WHERE empleadas_salarios.salario > empleadas_salarios.salario_medio
 GROUP BY pais;

```

Nota como una indentación correcta ayuda la legibilidad:

```
SELECT COUNT(id_empleada) AS NumeroEmpleadas,
       pais
  FROM (
    WITH salario_avg AS (
      SELECT pais,
             AVG(salario) AS salario_avg_pais
        FROM empleadas
       GROUP BY pais
    )
    SELECT e.id_empleada,
           e.nombre,
           e.apellido,
           e.pais,
           e.salario AS salario,
           s.salario_avg_pais AS salario_medio
      FROM empleadas AS e
     JOIN salario_avg AS s
       ON s.pais = e.pais
    ) AS empleadas_salarios
 WHERE empleadas_salarios.salario > empleadas_salarios.salario_medio
GROUP BY pais;
```

En este ejemplo hay que remarcar el hecho de que a los resultados devueltos por la CTE se le ha tenido que asignar un alias (empleadas_salarios) para poder acceder a sus columnas resultado desde la consulta externa. El resultado final de este ejercicio sería:

NumeroEmpleadas	pais
2	España

En los datos con los que estamos trabajando, de todos los países solo en España hay una empleada que cobre más que el salario medio de su país.

Justo antes del SELECT dentro de otras sentencias que contienen un SELECT

```
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
CREATE TABLE ... WITH ... SELECT ...
```

En estos casos, las CTEs se utilizarán de la manera que hemos visto a este momento, solo que su resultado será aplicado como entrada de consultas que no son del tipo `SELECT`, como por ejemplo un `INSERT`, `REPLACE` o `CREATE TABLE`.

Ejemplo: Podemos crear una tabla que contenga el nombre de cada país y el número de empleadas que cobran salarios por encima de la media de ese país. Es decir, el resultado del ejemplo anterior.

```
SELECT COUNT(id_empleada) AS NumeroEmpleadas,
       pais
  FROM (
    WITH salario_avg AS (
      SELECT pais,
             AVG(salario) AS salario_avg_pais
        FROM empleadas
       GROUP BY pais
    )
    SELECT e.id_empleada,
           e.nombre,
           e.apellido,
           e.pais,
           e.salario AS salario,
           s.salario_avg_pais AS salario_medio
      FROM empleadas AS e
     JOIN salario_avg AS s ON s.pais = e.pais
    ) AS empleadas_salarios
 WHERE empleadas_salarios.salario > empleadas_salarios.salario_medio
GROUP BY pais;
```

Esta es la consulta SQL con una CTE que hemos escrito en el ejemplo anterior. El resultado de la misma son dos columnas: NumeroEmpleadas y pais. A continuación introduciremos esta consulta dentro de un `CREATE TABLE` para guardar los datos en una tabla nueva:

```

CREATE TABLE EmpleadasSalarioAlto
( NumeroEmpleadas INT,
  pais VARCHAR(45)
);

INSERT INTO EmpleadasSalarioAlto
SELECT COUNT(id_empleada) AS NumeroEmpleadas,
       pais
  FROM (
    WITH salario_avg AS (
      SELECT pais,
             AVG(salario) AS salario_avg_pais
        FROM empleadas
       GROUP BY pais
    )
    SELECT e.id_empleada,
           e.nombre,
           e.apellido,
           e.pais,
           e.salario AS salario,
           s.salario_avg_pais AS salario_medio
      FROM empleadas AS e
     JOIN salario_avg AS s ON s.pais = e.pais
   ) AS empleadas_salarios
 WHERE empleadas_salarios.salario > empleadas_salarios.salario_medio
 GROUP BY pais;

```

El resultado será una tabla llamada EmpleadasSalarioAlto que estará formada por las dos columnas (con sus datos) resultado del `SELECT` que utiliza CTEs.

NumeroEmpleadas	pais
2	España

NOTA: Hay que remarcar que solo se permite la inclusión de una única cláusula `WITH` en cada nivel de un enunciado SQL. Es decir, un operador `WITH` seguido de otro `WITH` al mismo nivel *no es válido*. Si queremos definir más de una CTE en un nivel, se debe incluir una única vez el operador `WITH` para después separar las diferentes cláusulas mediante comas (cada CTE debe tener un nombre único dentro del operador `WITH`):

```

WITH cte1 AS (SELECT ...),
     cte2 AS (SELECT ...)
SELECT ...

```

Sin embargo, una sentencia SQL sí puede contener múltiples cláusulas `WITH` siempre que estas estén contenidas en diferentes niveles. Veámoslo con un ejemplo:

```

WITH cte1
AS (
    SELECT 1
)
SELECT *
FROM (
    WITH cte2 AS (
        SELECT 2
    )
    SELECT *
    FROM cte2
    JOIN cte1
) AS dt;

```

Por último, se debe tener en cuenta que una CTE también puede hacer referencia a otras CTEs de dentro de su misma cláusula `WITH` siempre y cuando estas hayan sido definidas con anterioridad (nunca después). Una CTE dentro de un bloque SQL también puede referenciar otras CTEs en niveles "superiores" de la consulta, pero no a aquellas CTEs definidas en niveles "inferiores".

```

WITH cte1 AS (SELECT ...),
      cte2 AS (SELECT ...
                FROM cte1)
SELECT ...

```

Tipos de CTE

Se puede distinguir entre dos tipos principales de CTEs según si estas utilizan algún tipo de recursividad en su definición.

CTE No Recursivas

Las CTE no recursivas son aquellas que no utilizan ningún tipo de recursividad en su definición. Estas son las CTEs estándar en las que como mucho se hacen referencia a otras CTEs dentro de la CTE actual pero nunca a sí misma.

Un ejemplo de este tipo de CTE sería el siguiente ejemplo:

```

WITH ContarFilas(NumeroFilas)
AS (
    SELECT ROW_NUMBER() OVER(
        ORDER BY nombre ASC
    ) AS NumeroFilas2
    FROM empleadas
    WHERE id_empleada <= 3
)
SELECT NumeroFilas
FROM ContarFilas;

```

En el ejemplo estamos definiendo una CTE llamada ContarFilas (redenominando la columna resultado como NumeroFilas) que contiene una consulta `SELECT`. Dicha consulta está usando la función `ROW_NUMBER`, la cual devuelve una secuencia de valores que comienza en 1 y se va incrementando en 1 en función de cuantas filas tenga el resultado original de la consulta. Sería algo similar a devolver una columna que indica el número de fila del resultado.

Dentro del `SELECT` de la CTE estamos consultando la tabla empleadas. Cada fila de esa tabla contiene atributos como el nombre de la empleada (`nombre`), un identificador para cada una (`id_empleada`), etc. Por lo tanto la consulta está eligiendo aquellas alumnas con un identificador menor de 3, las esta ordenando por su nombre (de manera ascendente) y está sacando el número de fila de cada registro del resultado.

Posteriormente, se incluye una consulta `SELECT` que hace uso del CTE ContarFilas incluyendo su nombre dentro del `FROM`. Como puede observarse, el nombre de la columna `NumeroFilas` se ha tomado del nombre que se definió en la CTE. De no haberse hecho de esa manera, los nombres de las columnas se deberían coger de la lista del `SELECT` dentro de la parte `AS` del CTE (en nuestro ejemplo sería `NumeroFilas2`):

```
WITH cte AS
(
    SELECT 1 AS col1, 2 AS col2
    UNION ALL
    SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

CTE Recursivas

Una CTE recursiva es aquella que se refiere a si misma en su definición. Por ejemplo:

```
WITH RECURSIVE cte (n)
AS (
    SELECT 1

    UNION ALL

    SELECT n + 1
    FROM cte
    WHERE n < 5
)
SELECT *
FROM cte;
```

En el ejemplo vemos la estructura típica de una CTE recursiva. La CTE debe comenzar con los operadores `WITH` y `RECURSIVE`, este último es siempre necesario si queremos definir una CTE recursiva aunque también se permite su uso en CTE no recursivas (aunque no tendrá ningún efecto). Posteriormente, la CTE recursiva tiene dos partes diferenciadas, separadas usando `UNION`.

El primer `SELECT` dentro de la CTE tiene el objetivo de producir un conjunto inicial de resultados que sirvan de entrada para posteriores ejecuciones de la CTE. En esta parte no se debe llamar a la propia CTE de manera recursiva. En nuestro ejemplo concreto esta primera parte simplemente estaría devolviendo el número 1 al ejecutarse.

Después, el segundo `SELECT` (el que está después del `UNION`) utiliza en primer lugar los resultados del `SELECT` anterior y va produciendo resultados adicionales que va recibiendo como entrada durante las etapas de recursividad. La recursividad de la CTE terminará cuando este segundo `SELECT` ya no produzca nuevas filas resultado. Debido a esta condición, si queremos asegurarnos de que la recursividad termine en algún momento, siempre debemos proporcionar una condición `WHERE` que haga que se termine la ejecución. En nuestro ejemplo será que el resultado recibido sea menor que 5.

En el ejemplo que hemos mostrado, la parte recursiva coge el resultado devuelto por la anterior llamada y le suma 1. La llamada terminará cuando se deje de cumplir la condición impuesta en el `WHERE` ya que entonces no se producirán nuevos resultados.

El orden de ejecución de la CTE al llamarla sería el siguiente. En una primera instancia, `SELECT 1` devuelve un número 1 y luego pasa a ejecutarse el segundo `SELECT` que recibe ese 1 y le va sumando 1 cada vez que se realiza la recursividad. Irá sumando y sumando hasta que el valor recibido vale 5, cuando ya no se cumpliría el `WHERE` y por lo tanto no se devolverían nuevos resultados, acabando la ejecución. La tabla resultado de la ejecución completa sería la siguiente:

n
1
2
3
4
5

Resumiendo todo lo anterior, una CTE recursiva consiste en una primera parte con un `SELECT` no recursivo y un segundo `SELECT` donde se utiliza la recursividad.

Restricciones para las CTE recursivas

En las CTEs no se pueden utilizar todas las funciones y operadores de SQL que hemos estudiado hasta el momento. Algunas de las restricciones que existen para *la parte recursiva* de las CTEs son:

- No se pueden usar funciones agregadas como `SUM()`, `MAX()`, etc.
- No se pueden usar funciones ventana.
- No se pueden usar `GROUP BY`, `ORDER BY`, ni `DISTINCT`.

Todas estas restricciones solo se aplican a la parte recursiva de las CTEs, el `SELECT` no recursivo no tiene ninguna restricción.

Por otro lado, la parte recursiva de la CTE debe referenciarse a sí misma una vez (y solo una) dentro de la cláusula `FROM`, nunca dentro de otra subconsulta. También puede tener referencias a otras tablas que no sean de la propia CTE y realizar cualquier `JOIN` entre ellas y la propia CTE. Sin embargo, no se puede realizar un `LEFT JOIN` donde la CTE esté en el lado derecho.

EJERCICIO 1

Las CTE pueden ser útiles para recorrer datos que siguen cierta jerarquía, como podría ser la cadena de supervisión dentro de una empresa. Imaginemos que en esa empresa cada empleada tiene una supervisora. Esa supervisora a su vez tiene una supervisora y así sucesivamente hasta llegar a la "cabeza" de la jerarquía, que podría ser la CEO de la empresa. La tabla `empleadas` que contiene esta información en la base de datos tendría el siguiente formato:

<code>id_empleada</code>	nombre	<code>id_supervisora</code>
29	Pedro	198
72	Paco	29
123	Alba	692
198	Laura	333
333	Yasmina	NULL
692	Tomas	333
4610	Sara	29

Escribe una CTE recursiva que muestre cada empleada (su `id_empleada`), su nombre, y el camino desde la CEO hasta ese propio empleado, en forma de lista de ids separados por comas. Para la tabla ejemplo que hemos mostrado anteriormente, la salida que queremos que devuelva la CTE cuando la usemos sería algo como lo siguiente:

id_empleada	nombre	camino
333	Yasmina	333
198	Laura	333,198
29	Pedro	333,198,29
4610	Sara	333,198,29,4610
72	Paco	333,198,29,72
692	Tomas	333,692
123	Alba	333,692,123

PISTA: empieza sacando el id_empleada de la CEO en la parte no recursiva de la CTE.

AYUDA: Para crear las tablas necesarias:

```

CREATE TABLE empleadas_camino(
    id_empleada INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(30) DEFAULT NULL,
    id_supervisora INT DEFAULT NULL,
    PRIMARY KEY (id_empleada)
);

INSERT INTO empleadas_camino
VALUE (29,'Pedro',      198),
(72,'Paco',       29),
(123,'Alba',      692),
(198,'Laura',     333),
(333,'Yasmina',   NULL),
(692,'Tomas',     333),
(4610,'Sara',     29);

```

Resumen CTE's

- CTE: Conjunto de resultados temporales con nombre propio(obligatorio), que pueden ser referenciadas por otra instrucción SQL.
 - Ventajas de las CTE:
 - Reflejan la lógica humana a la hora de escribir las consultas.
 - Ayuda a que las consultas sean más legibles
 - Permite organizar de forma más clara las consultas largas
 - Existe un tipo especial de CTE llamadas CTE recursivas que permiten una autoreferencia (similar a los bucles for)
 - Sintaxis de las CTE's:

```
WITH nombre_cte (nombre_columnas) AS ( consulta sql a realizar)
```

- Posiciones de las consultas CTE:
 - Antes de una consulta de tipo SELECT, UPDATE, DELETE
 - Al comienzo de subconsultas dentro de otra consulta SELECT
 - Antes de un SELECT dentro de otras sentencias que tienen un SELECT.
- Podemos utilizar varias CTE's:
 - Poniendo una detrás de otra, separándolas por comas.-
 - Anidándolas a diferentes niveles.
- Tipos de CTE's
 - CTE's no recursivas: Hacen referencia a otras CTE's o consultas/subconsultas, pero no a si misma. No podemos ejecutar la CTE de forma independiente, es decir, siempre necesita de una consulta en la que se hace referencia.
 - CTE's recursivas: Hacen referencia a si mismas, lo cual hace que se puedan ejecutar de forma independiente.
 - Necesitamos especificar un WHERE con la condición de salida de la CTE recursiva.

Pair programming Simplificando consultas con CTEs

Recordad que no es necesario que hagas todos los ejercicios de este módulo, son muchos! Lo importante es que vayáis entendiendo cada query que hagáis

Enunciado

Hoy practicaremos las CTE's .

Actividades

- Extraer en una CTE todos los nombres de las compañías y los id de los clientes.

Para empezar nos han mandado hacer una CTE muy sencilla el id del cliente y el nombre de la compañía de la tabla Customers.

Los resultados de esta query serán:

#	CustID	CompanyName
1	ALFKI	Alfreds Futterkiste
2	ANSTR	Ana Trujillo Emparedados y helados
3	ANTON	Antonio Moreno Taquería
4	AROUT	Around the Horn
5	BSBEV	B's Beverages
6	BOLID	B?lido Comidas preparadas
7	BERGS	Berglunds snabbköp
8	BLAUS	Blauer See Delikatessen
9	BLONP	Blondesddsl p?re et fils
10	BONAP	Bon app'
11	BOTTM	Bottom-Dollar Markets
12	CACTU	Cactus Comidas para llevar

sql101

- Selecciona solo los de que vengan de "Germany"

Ampliemos un poco la query anterior. En este caso, queremos un resultado similar al anterior, pero solo queremos los que pertenezcan a "Germany".

Los resultados de esta query serán:

#	CustID	CompanyName
1	ALFKI	Alfreds Futterkiste
2	BLAUS	Blauer See Delikatessen
3	DRACD	Drachenblut Delikatessen
4	FRANK	Frankenversand
5	KOENE	K?niglich Essen
6	LEHMS	Lehmanns Marktstand
7	MORGK	Morgenstern Gesundkost
8	OTTIK	Ottilie K?seladen
9	QUICK	QUICK-Stop
10	TOMSP	Toms Spezialit?ten
11	WANDK	Die Wandernde Kuh

sql101

- Extraed el id de las facturas y su fecha de cada cliente.

En este caso queremos extraer todas las facturas que se han emitido a un cliente, su fecha la compañía a la que pertenece.

NOTA En este caso tendremos columnas con elementos repetidos(*CustomerID*, y *Company Name*).

Los resultados de esta query serán:

#	customer_id	company_name	order_id	order_date
1	ALFKI	Alfreds Futterkiste	10643	1997-08-25 00:00:00
2	ALFKI	Alfreds Futterkiste	10692	1997-10-03 00:00:00
3	ALFKI	Alfreds Futterkiste	10702	1997-10-13 00:00:00
4	ALFKI	Alfreds Futterkiste	10835	1998-01-15 00:00:00
5	ALFKI	Alfreds Futterkiste	10952	1998-03-16 00:00:00
6	ALFKI	Alfreds Futterkiste	11011	1998-04-09 00:00:00
7	ANSTR	Ana Trujillo Emparedados y helados	10308	1996-09-18 00:00:00
8	ANSTR	Ana Trujillo Emparedados y helados	10625	1997-08-08 00:00:00
9	ANSTR	Ana Trujillo Emparedados y helados	10759	1997-11-28 00:00:00
10	ANSTR	Ana Trujillo Emparedados y helados	10926	1998-03-04 00:00:00

4. Contad el número de facturas por cliente

Mejoremos la *query* anterior. En este caso queremos saber el número de facturas emitidas por cada cliente.

Los resultados de esta *query* serán:

#	customer_id	company_name	numero_facturas
1	ALFKI	Alfreds Futterkiste	6
2	ANATR	Ana Trujillo Emparedados y helados	4
3	ANTON	Antonio Moreno Taquería	7
4	AROUT	Around the Horn	13
5	BERGS	Berglunds snabbköp	18
6	BLAUS	Blauer See Delikatessen	7
7	BLONP	Blondesddsl p?re et fils	11
8	BOLID	B?lido Comidas preparadas	3
9	BONAP	Bon app'	17
10	BOTTM	Bottom-Dollar Markets	14

5. Cuál la cantidad media pedida de todos los productos ProductID.

Necesitaréis extraer la suma de las cantidades por cada producto y calcular la media.

Los resultados de esta *query* serán:

#	producto	media_
1	Alice Mutton	32.5909
2	Aniseed Syrup	28.6667
3	Boston Crab Meat	26.5500
4	Camembert Pierrot	35.6000
5	Carnarvon Tigers	20.8000
6	Chai	24.4286
7	Chang	27.4211
8	Chartreuse verte	32.3158
9	Chef Anton's Cajun Seasoning	19.5000
10	Chef Anton's Gumbo Mix	31.0000

Happy coding

Repaso SQL I

Aquí podéis encontrar el fichero csv completo utilizado a lo largo de la primera sesión de repaso del módulo 1.



datos_spotify.csv 42KB

Binary

El archivo que vamos a usar para este ejercicio.



repaso1.sql 2KB

Binary

El archivo que vamos a usar para este ejercicio.

Repaso SQL II

Día de repaso SQL 2

Aquí podéis encontrar el fichero SQL completo utilizado a lo largo de la segunda sesión de repaso del módulo 1.



table_season.csv 147B
Binary



table_teams.csv 996B
Binary



team_season_stats_tables.csv 25KB
Binary

El archivo que vamos a usar para este ejercicio.

Explicación de las variables de los datasets

Tabla Seasons

SEASON_ID: Identificador único de temporada

Season: Número de temporada

Tabla Teams

TEAM_ID: Identificador único de cada equipo

TEAM: Nombre del equipo (lo utilizaremos para crear el ID único de cada equipo)

Tabla stats

TEAM_STAT_SEASON_ID: Identificador único de cada registro de las estadísticas por partido

SEASON_ID: Identificador único de temporada

TEAM_ID: Identificador único de temporada

GP (Games Played): Número de partidos que ha jugado dicho equipo en esa temporada.

W (WINS): Número de partidos que ha ganado dicho equipo.

L (LOSS): Número de partidos que ha perdido dicho equipo.

WIN% (WIN PERCENTAGE): Porcentaje de partidos ganados. Cociente de partidos ganados, entre partidos totales.

MIN (MINUTES): Número de minutos medio por temporada.

PTS (POINTS): Número de puntos.

FGM (FIELD GOALS MADE): Número de canastas encestadas.

FGA (FIELD GOALS ATTEMPTED): Número de veces que se ha tirado a canasta.

FG% (FIELD GOALS %): Porcentaje del cociente de canastas encestadas, entre el número de veces que se ha tirado a canasta.

3PM (THREE POINTS MADE): Número de veces que se han metido canastas de 3 puntos.

3PA (THREE POINTS ATTEMPTED): Número de veces que se han intentado meter canastas de 3 puntos.

3P% (THREE POINTS PERCENTAJE): Porcentaje de canastas de 3 puntos. Cociente de 3PM entre 3PA.

FTM (FREE THROWS MADE): Número de tiros libres realizados.

FTA (FREE THROWS ATTEMPTED): Número de tiros libres que se han intentado.

FT% (FREE THROWS PERCENTAGE): Porcentaje de tiros libres. Cociente de FTM entre FTA.

OREB (OFFENSIVE REBOUNDS): Número de rebotes ofensivos.

DREB (DEFENSIVE REBOUNDS): Número de rebotes defensivos.

REB (REBOUNDS): Número total de rebotes. Debería ser la suma de OREB y DREB.

AST (ASSISTS): Número de asistencias.

TOV (TURNOVERS): Perdida de balón.

STL (STEALS): Número de veces que se ha robado el balón.

BLK (BLOCKS): Número de veces que se ha bloqueado.

BLKA (BLOCKED FIELD GOAL ATTEMPTS): Número de veces que se ha bloqueado una tirada a canasta.

PF (PERSONAL FOUL): Faltas.

PFD (PERSONAL FOULS DRAWN): Faltas sacadas.

+/- (PLUS MINUS): Diferencia de puntos cuando un jugador o equipo está en la cancha.

Ficheros de la resolución

 repaso2.sql	9KB
Binary	

Repaso SQL III

Día de repaso SQL 3

Aquí podéis encontrar el fichero SQL completo utilizado a lo largo de la tercera sesión de repaso del módulo 1.

Ficheros de la resolución

Orden de escritura	Orden de ejecución
SELECT	FROM
DISTINCT	JOIN
FROM	WHERE
JOINS	GROUP BY
WHERE	HAVING
GROUP BY	SELECT
HAVING	DISTINCT
ORDER BY	ORDER BY
LIMIT - OFFSET	LIMIT - OFFSET

Tablas orden de ejecución



repaso3.sql 6KB
Binary

Repaso SQL IV

Día de repaso SQL 4

Aquí podéis encontrar el fichero SQL completo utilizado a lo largo de la cuarta sesión de repaso del módulo 1.

Ficheros de la resolución



repaso4.sql 5KB

Binary



intento_funcion_sql_repaso4.sql 846B

Binary

Proyecto 1

Nuestro cliente, Adalab, está inmersa en un proceso activo de transformación digital y desea testear nuestras capacidades de analistas de datos.

A raíz de esto nos ha mandado una serie de diferentes ficheros que contienen información relacionada entre sí y desea obtener una base de datos que agregue toda la información en diferentes tablas.

De esta forma nos ha mandado 3 ficheros:

- Archivo en formato XML.
- Archivo en formato txt.
- Archivo en formato sql.

Como rol de futuras analistas de datos, os ha pedido que las tablas finales tengan la información parcialmente procesada. Y que automaticéis todo el sistema de procesado de datos, ya que de forma periódica se recibirán otras remesas de datos similares actualizados.

Índice

- Resumen
- Objetivos
- Caso de uso
- Especificaciones
- Planificación del proyecto
 - Sprints
 - Historias de usuario
- Entrega
- Presentación

Resumen

En este proyecto vamos a aprender a tratar una serie de archivos relacionados en diferentes formatos, concretamente texto, xml y de una base de datos. Para ello vamos a tener que ser capaz de abrir los ficheros, procesarlos y realizar algunas transformaciones sencillas sobre los datos recibidos. ¡Esta será vuestra primera experiencia de trabajo en equipo relacionada con programación! ¿Estáis preparadas?

Objetivos

1. Consolidar los conocimientos de Python básicos, así como el tratamiento de ficheros en diversos formatos y extracción de datos de una base de datos.
2. Procesar y modificar las variables de tipo texto, para que sea más fácil trabajar con estos datos en futuros proyectos.
3. Utilizar control de versiones en equipo para aprender las ventajas y conflictos que genera.
4. Implementar Scrum como marco de referencia para el desarrollo del producto, basándonos siempre en los valores de Agile como puntos clave del trabajo en equipo y la mejora continua.
5. Mejorar la comunicación entre los miembros del equipo.
6. Mejorar vuestras habilidades de comunicación en público al exponer el proyecto en la sesión final.

Caso de uso

Con este proyecto vais a demostrar, que datos una serie de conjuntos de datos en diferentes ficheros y formatos, sois capaces de procesarlos, corregir algunos valores erróneos de las variables de tipo texto. Esto os permitirá mostrar vuestras habilidades de enfrentarnos a un conjunto de datos desde cero en GitHub. Algo que os será útil a la hora de demostrar vuestros conocimientos a las empresas durante los futuros procesos de selección a los que os enfrentais.

Stack tecnológico

En desarrollo del procesado del conjunto de datos haremos uso de las siguientes tecnologías y paquetes de Python:

- Extracción y guardado de datos desde una base de datos en MySQL
- Procesado de datos haciendo uso de :
 - `xml.etree.ElementTree`
 - Librerías base de Python
- Ficheros en formato:
 - `txt`
 - `xml`
 - `sql`

Especificaciones

Se desarrollará una base de datos con las siguientes características:

- Uso de MySQL (Al principio se trabajará sobre MySQL Workbench, pero al final se realizará todo el trabajo desde Python)
- Se debe tener creada la infraestructura necesaria: repositorio en GitHub y con acceso para todos los miembros del equipo. Siguiendo una estructura de carpetas coherente y sencilla. **Para considerar terminada las diferentes historias de usuarias del proyecto debéis tener publicada las soluciones en el repositorio de GitHub.**
- Elaboración de una presentación para el dia de la demo.

A los diferentes ficheros recibidos habrá que hacerle una serie de transformaciones con el fin de dejar los datos parcialmente procesados. Algunas de las transformaciones esperadas son:

- Fichero SQL:
 - Contiene el campo ERROR en lugar de NULL. Por lo tanto deberemos modificarlo.
- Fichero XML:
 - Contiene algunas columnas redundantes.
 - La variable género esta codificada y deberemos cambiarla:

gender	gender_text
0	Man
1	Woman
2	Nonbinary
3	Prefer not to say
4	Prefer to self-describe

- Fichero TXT:
 - Contiene espacios extra en algunos strings, que deben ser eliminados
 - Contiene elementos como '\n', que deben ser eliminados
 - Contiene elementos como '<', que deben ser cambiados por 'under'
 - Contiene elementos como null, que deben ser cambiados por 'NULL'

NOTA: Se pueden realizar procesados adicionales a los aquí indicados si se desea.

La tabla de la base de datos de MySQL contendrá las siguientes tablas:

- Tabla SQL: Contendrá los datos en formato SQL que nos ha proporcionado el cliente.
 - Deberá tener una clave primaria y será la tabla madre del resto de tablas. La clave primaria será de tipo numérico, el resto de columnas será de tipo texto.

La tabla final contendrá las siguientes columnas:

columnas

index_sql

q10_part_1

q10_part_2

q10_part_3

q10_part_4

q10_part_5

q10_part_6

q10_part_7

q10_part_8

q10_part_9

q10_part_10

q10_part_11

q10_part_12

q10_part_13

q10_part_14

q10_part_15

q10_part_16

q10_other

d482xta

Podemos ver en la siguiente imagen el resultado final esperado:

id	id_sql	id_part_1	id_part_2	id_part_3	id_part_4	id_part_5	id_part_6	id_part_7	id_part_8	id_part_9	id_part_10	id_part_11	id_part_12	id_part_13	id_part_14	id_part_15	id_part_16	id_other
1	Kaggle Notebooks	Codelab notebooks	80%															
2	Kaggle Notebooks	Codelab notebooks	80%															
3																		
4																		
5																		
6																		
7																		
8																		
9																		
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		
21																		
22																		
23																		
24																		

- Tabla XML: Contendrá los datos en formato xml que nos ha proporcionado el cliente.

- Deberá tener una clave primaria y tendrá una clave foránea referenciado a la tabla SQL.

La clave primaria y foránea serán de tipo numérico, el resto será de tipo texto.

La tabla final contendrá las siguientes columnas:

columnas

index_xml

time

age

gender

index_sql

Podemos ver en la siguiente imagen el resultado final esperado:

	index_xml	time	age	gender	index_sql
1		784	50-54	Man	1
2		924	22-24	Man	2
3		575	45-49	Man	3
4		781	45-49	Man	4
5		1020	25-29	Wo...	5
6		141	18-21	Wo...	6
7		484	30-34	Man	7
8		1744	22-24	Man	8
9		655	30-34	Man	9
10		1777	40-44	Man	10
11		3081	18-21	Wo...	11
12		1922	18-21	Wo...	12
13		852	45-49	Man	13
14		838	22-24	Man	14

- Tabla txt: Contendrá los datos en formato xml que nos ha proporcionado el cliente.

- Deberá tener una clave primaria y tendrá una clave foránea referenciado a la tabla SQL. La clave primaria y foránea serán de tipo numérico, el resto serán de tipo texto. La tabla final contendrá las siguientes columnas:

columnas

index_txt

index_sql

q3

q4

q5

q6

q7

q8

q9

q11

q12

q13

q14

q15

q16

q17

q20

q21

q22

q23

q24

q25

q26

q32

q33

q34

q35

q41

Podemos ver en la siguiente imagen parte del resultado final esperado:

Index	First	Last	Education	Experience	Skills	Software	Cloud	Hardware	Other
1	John	Doe	Master's degree	Program/Project Manager	2-5 years	N/A	N/A	N/A	Jupyter Notebook, Python, Java
2	John	Doe	Possess - Master's degree	Software Engineer	2-5 years	Python, C++, Java	PyCharm, N/A	N/A	Jupyter Notebook, Python, Java
3	John	Doe	Master's degree	Research Scientist	2-5 years	Python	PyCharm, Jupyter Notebook	A laptop	N/A
4	John	Doe	Doctoral degree	Data Scientist	3-5 years	Python, R, MATLAB	PyCharm, Jupyter Notebook	A cloud computing platform (AWS, ...)	A laptop
5	John	Doe	Other	Data Scientist	Under 1 year	Python	PyCharm, Jupyter Notebook	A cloud computing platform (AWS, ...)	A laptop
6	John	Doe	Other	Current not employed	Under 1 year	Python	PyCharm, Jupyter Notebook	A laptop	N/A
7	John	Doe	Bachelor's degree	Data Scientist	5-10 years	Python	PyCharm, Jupyter Notebook	A personal computer / desktop	N/A
8	John	Doe	Bachelor's degree	Data Scientist	5-10 years	Python, SQL	PyCharm, Jupyter Notebook	A cloud computing platform (AWS, ...)	N/A
9	John	Doe	Bachelor's degree	Other	1-3 years	Python, R, SQL	R, PyCharm, Jupyter Notebook	A personal computer / desktop	N/A
10	John	Doe	Bachelor's degree	Other	1-3 years	Python, R, SQL	R, PyCharm, Jupyter Notebook	A personal computer / desktop	N/A
11	John	Doe	Doctoral degree	Student	Under 1 year	Python, MATLAB	PyCharm, Jupyter Notebook	A laptop	N/A
12	John	Doe	Master's degree	Student	Under 1 year	Python, MATLAB	PyCharm, Jupyter Notebook	A laptop	N/A
13	John	Doe	Master's degree	Program/Project Manager	5-10 years	Python, SQL	PyCharm, Jupyter Notebook	A laptop	N/A
14	John	Doe	Other	Other	Under 1 year	Python	PyCharm, Jupyter Notebook	A laptop	N/A

Planificación del proyecto

Sprints

Para la realización de este proyecto trabajaremos en 2 sprints (2 iteraciones) el primer sprint tendrá una duración de 8 sesiones y el segundo sprint tendrá una duración de 6 sesiones. Siguiendo los principios ágiles, estableceremos pequeños ciclos iterativos de forma que al final de cada uno generaremos valor perceptible por nuestros usuarios (los visitantes de la web). Dedicaremos el primer día a la planificación del sprint (sprint planning) y el resto a trabajar en el desarrollo del proyecto. Al final de cada sprint haremos un Sprint Review (demo) del proyecto para presentar los resultados conseguidos y recoger feedback, al igual que una retrospectiva (retro) para evaluar cómo ha ido el sprint, además de valorar vuestro trabajo en equipo de cara a mejorar en el siguiente sprint.

También haremos una retro corta revisando los working agreements que hemos acordado al inicio del proyecto y añadiendo cualquier otro feedback que nos permita mejorar el proyecto.

Al final del sprint (que coincidirá con el final del proyecto), haremos una sesión de presentación más completa, más allá de lo que sería un Sprint Review

Historias de usuaria

Para la gestión del proyecto, usaremos historias de usuaria, que es una herramienta para definir las características de un producto que veremos en detalle durante el curso.

Primera. Creación de la base de datos, inserción, modificación y extracción los ficheros de la base de datos a fichero externo.

Valor

El cliente quiere disponer de toda la información en diferentes tablas y contenida en una base de datos para agregarla a los datos ya existentes.

Contexto

Con la transformación digital que esta siguiendo Adalab, desea tener un sistema de BBDD donde poder alojar toda la información que se reciba de forma periódica. Ya que utilizar esta clase de sistemas les permitirá tener redundancia de los datos, que estén seguros y que múltiples personas puedan tener acceso a todos los datos que nos ha trasmido la empresa.

Criterios de aceptación

- Tener una base de datos.
- La base de datos contiene 3 tablas diferentes con la información de los datos recibidos en xml,txt y sql.
- Las diferentes tablas deben estar conectadas entre si mediante claves primarias y foráneas.
- Insertar la información de las diferentes tablas.
- **DOD: Se han exportado los datos de MySQL un fichero externo tipo sql**

Segunda.Limpieza de los datos de los ficheros xml automatizada.

Valor

El cliente quiere tener los datos recibidos del fichero xml limpios todos los procesos de limpieza lo más automatizados posible, para ello se implementará los métodos más comunes de tratamiento de strings.

Contexto

De nuevo, debido a que se espera un flujo de datos nuevos periódicos, se busca ser capaces de realizar una sencilla limpieza de datos, considerando los casos más comunes a utilizar para la validación correcta de los datos.

Criterios de aceptación

- Leer el fichero xml
- Procesar los datos contenidos en este fichero.
- Crear funciones para limpiar este tipo de datos de forma automática.
- **DOD: Tener los datos procesados del fichero xml en un diccionario de Python.**

Tercera.Limpieza de los datos de los ficheros txt automatizada.

Valor

El cliente quiere tener los datos recibidos del fichero txt limpios todos los procesos de limpieza lo más automatizados posible, para ello se implementará los métodos más comunes de tratamiento de strings.

Contexto

De nuevo, debido a que se espera un flujo de datos nuevos periódicos, se busca ser capaces de realizar una sencilla limpieza de datos, considerando los casos más comunes a utilizar para la validación correcta de los datos.

Criterios de aceptación

- Leer el fichero txt
- Procesar los datos contenidos en este fichero.
- Crear funciones para limpiar este tipo de datos de forma automática.
- **DOD: Tener los datos procesados del fichero txt en un diccionario de Python.**

Cuarta. Inserción de los datos de los ficheros xml y txt a la BBDD mediante Python.

Valor

El cliente quiere poder insertar la información haciendo uso del lenguaje de programación Python.

Contexto

El cliente ha tenido una reunión con el equipo de datos de la empresa y ha descubierto que en su equipo existen algunas analistas de datos que saben programar el Python, por lo tanto desea poder insertar los datos desde Python en la base de datos SQL existente.

Criterios de aceptación:

- Crear el código para la creación de tablas SQL desde Python mediante el paquete mysql.connector.
- Crear el código para la inserción de los datos desde Python a las tablas de SQL.
- **DOD: Los datos de los ficheros txt y xml procesados deben estar insertados en la base de datos de MySQL**

Quinta. Automatización de lectura de archivos, procesado y actualización de datos.

Valor

El cliente quiere tener todos los procesos de lectura de ficheros, tratamiento e inserción de datos automatizados.

Contexto

El cliente ha tenido una reunión con el equipo de datos de la empresa y se tiene la previsión de que la carga de los ficheros, la validación de los datos y su procesado deba realizarse de forma periódica. Con una estimación de repetir el proceso cada dos semanas, por lo tanto hacer el proceso manualmente cada dos semanas no resulta eficiente en términos de tiempo de trabajo.

Criterios de aceptación:

- Creación de clase con las funciones de lectura de datos
- Creación de clase con las funciones de procesado de datos.
- Creación de clase con las funciones de creación de tablas e inserción de datos desde Python a la base de datos SQL .
- **DOD: Tener una varias clases que engloben los diferentes procesos aplicados a los datos recibidos.**

Ficheros

Los ficheros a descargar son los siguientes:



[data_sql.sql](#) 4MB

Binary

[Descarga este fichero.](#)



[data_txt.txt](#) 14MB

Binary

[Descarga este fichero.](#)



[data_xml.xml](#) 3MB

Binary

[Descarga este fichero.](#)

El fichero para poder leer los datos del fichero xml y txt. Así como algunas recomendaciones:



[read_xml_txt.ipynb](#) 5KB

Binary

[Descarga este fichero.](#)

Entrega

El formato de entrega de este proyecto será mediante la subida de este a la plataforma de GitHub. Para subirlo, se creará un repositorio en la organización de Adalab. El nombre del repositorio deberá estar compuesto de las siguientes partes, todo ello separado por guiones:

- La palabra project-da.
- Letra de la promoción promo-B.
- Número del módulo module-1.
- Número del equipo team-X. Por ejemplo:
- Adalab/project-da-promo-B-module-1-team-1
- Adalab/project-da-promo-B-module-1-team-3

En lo relacionado en las fechas de los sprints:

- Entrega del primer sprint (sprint review): 18 Noviembre.
- Entrega del segundo sprint (sprint review): 28 de Noviembre.
- Demo del proyecto (presentación final y retro): 30 de Noviembre

En las sprint review se revisarán que se hayan solucionado todas las tareas técnicas asociadas a la entrega de esos sprints, si algo quedara pendiente se arrastraría al siguiente sprint.

Presentación

El último día del módulo presentaréis la versión final de este proyecto a vuestras compañeras y al equipo de Adalab. Cada equipo realizará una presentación de 5 minutos y posteriormente habrá 5 minutos de feedback por parte del público. En este caso, la audiencia podría ser más variada pues no sólo estarán los profesores.

El objetivo es que practiquéis la realización de las demos de los proyectos que habéis desarrollado, explicándolo desde un punto de vista técnica y también desde la perspectiva del producto, mejorando además vuestras habilidades de exposición, objetivo de desarrollo profesional del curso.

Para que la presentación salga bien es imprescindible una buena preparación. Por ello, durante el segundo sprint del módulo tendréis que asignar responsabilidades dentro del equipo relacionadas con la preparación de ésta. A continuación incluimos algunos elementos que os pueden ayudar a enfocar la presentación:

- En el público habrá personas con conocimientos técnicos y no técnicos.
- La parte central de la presentación será mostrar el software desarrollado funcionando, a ser posible en directo de forma dinámica o a través de un vídeo (si no fuera posible, como plan B).
- En este módulo, de los diferentes elementos adicionales que os proponemos, sería útil incluir una breve presentación de los diferentes integrantes del equipo desde un punto de vista profesional. Se trata de practicar vuestro "relato" profesional en versión muy corta. Que las personas asistentes conozcan quienes sois como profesionales. Os será también útil para las entrevistas de trabajo.
- Todas las participantes del equipo deben hablar en la presentación (sin práctica no hay mejora).

Además de esto, para mejorar vuestras habilidades de exposición en público y hacer la presentación más rica, podréis incorporar otros elementos adicionales (son solo ideas, sentíos libres de innovar y ser creativas):

- Dejar muy claro quién ha sido vuestro cliente y qué fue lo que os pidió.
- Explicar qué necesidades cubre o qué problemas soluciona el producto, cuál es el beneficio principal que aporta y qué lo hace único comparado con otros productos parecidos del mercado.
- Aportaciones "únicas y diferenciadoras" de cada equipo al proyecto.
- Cómo ha sido la organización del equipo, el reparto de tareas y la coordinación a la hora de trabajar todas en el mismo código.
- Cuál de las tareas o los puntos ha sido el que más esfuerzo ha requerido.
- Cuál de las tareas o partes de la web es la que hace que el equipo esté más orgulloso.
- Las tecnologías qué habéis utilizado y para qué sirven, y algunas partes del código que habéis desarrollado que merezca la pena resaltar.
- La presentación debe tener un "buen inicio y un buen cierre" que nos haga a todos estar atentos y aplaudir... ahí os dejamos que echéis a volar vuestra imaginación.
- No habléis en primera persona de lo que habéis hecho, hablad del equipo.
- No mencionéis problemas, sino "retos" que os han hecho aprender y crecer.

MODULO 2: Limpieza y visualización de datos con Python

Numpy

NumPy I - Introducción a NumPy

¿Qué es NumPy? Entenderemos que es un array o matriz, cómo podemos crearlas y qué tipo de operaciones podemos realizar.

NumPy es, al igual que Pandas, es uno de los paquetes que no puedes perderte cuando estás aprendiendo análisis de datos, principalmente porque esta librería proporciona una estructura de datos de matriz que tiene algunos beneficios sobre las listas regulares de Python.

NumPy es un paquete de Python que significa “Numerical Python”, es la librería principal para la informática científica, proporciona potentes estructuras de datos, implementando matrices y matrices multidimensionales. Estas estructuras de datos garantizan cálculos eficientes con matrices.



Numpy-I.ipynb 270KB
Binary

[Descarga este Jupyter y ábrelo en VS Code.](#)

Reparo

- NumPy es una librería especializada para el cálculo numérico.
- Utiliza los `array` como estructura de datos.
- Solo permite datos de un mismo tipo. Ya sea: Bool, Int, Float, str ...
- Puede tener múltiples dimensiones:
 - 1D
 - 2D
 - 3D
 - N-D
- Podemos crear `arrays` utilizando listas.
- Para operar entre `arrays` necesitamos que ambos contengan los mismos tipos de datos.
 - En sumas y restas han de tener las mismas dimensiones
 - En multiplicación y división se debe cumplir la regla de multiplicación de matrices.
- Podemos realizar operaciones entre `arrays` y escalares(siendo un escalar un número)
- Creación de `arrays`:
 - Podemos tener `arrays` de diferentes dimensiones:
 - Una dimensión : `np.array(lista)`
 - Dos dimensiones : `np.array(lista1, lista)`
 - Tres dimensiones: `np.array(lista1(lista), lista2(lista))`
- Métodos de `arrays`:
 - `.shape` : Nos devuelve la forma del `array`.
 - En un `array` bidimensional nos devuelve: numero filas X numero de columnas
 - En un `array` multidimensionales: numero de matrices X numero de filas X numero de columnas
 - `.size` : Nos devuelve el tamaño del `array`, que es el número de elementos.
 - `.ndim` : Nos devuelve el número de dimensiones del `array`.
 - `.dtype` : Nos devuelve el tipo de dato que contiene nuestro `array`.
- Métodos de creación de `arrays`:
 - `np.random.randint()` : Nos genera números aleatorios enteros
 - `np.random.rand()` : nos genera números aleatorios decimales (`float`) desde el 0-1 (incluido)
 - `np.random.random_sample()` : nos genera números decimales (`float`) entre 0 -1 (NO incluido)
 - `np.ones()` nos permite crear matrices de todo 1
 - `np.zeros()` nos permite crear matrices de todo 0
 - `np.ones_like()` , `np.zeros_like()` : nos permite crear matrices de 0 y 1 respectivamente con la forma de otra matriz definida previamente.
 - `np.eye()` : es como una matriz identidad, pero le podemos modificar la posición de la diagonal
 - `np.identity()` que nos permite crear una matriz identidad, todo 0 menos la diagonal que son

1.

- `np.array()` : nos permite crear matrices a partir de una lista
- `np.empty()` : nos crea array vacío. Pero ojo! Nunca será vacío como tal, nos creará los valores en función de la matriz anterior.
- Podemos hacer operaciones en NumPy:
 - Podemos sumar con: `+` o `np.add()`
 - Restar con: `-` , `np.subtract()`
 - Multiplicar con: `*` , `np.multiply()`
 - Dividir con: `/` , `np.divide()`
 - Podemos hacer todas las operaciones que queramos!!!!
 - Podemos hacer operaciones con escalares, números enteros o decimales.

Pair Programming NumPy I

Empezamos nuevo bloque de ejercicios de *pair programming* de NumPy.

En el ejercicio de hoy vamos a empezar a trabajar con NumPy. El objetivo de estos ejercicios es que creeis una serie de *arrays*, y hagáis una serie de operaciones matemáticas con ellos.

1. Cread tres *arrays*: de una, dos y tres dimensiones.

Lo podéis hacer usando los métodos de `random` que hemos aprendido o a través de listas en algunos casos.

Nota

- El *array* de 2 dimensiones debe ser de 2 filas y 3 columnas
- El *array* de 3 dimensiones debe ser de 2 matrices, 3 filas y 5 columnas

2. Chequed las las propiedades básicas de cada *array* que os hayáis creado. Usad en una función que nos devuelva las propiedades de una *array* que le pasemos.
3. Cread una matriz identidad de dos dimensiones de 3 filas y 3 columnas
4. Cread una matriz de tres dimensiones de unos igual a la matriz de tres dimensiones creada en el primer ejercicio.
5. Es el momento de hacer algunas operaciones entre arrays
 - ¿Se puede sumar el array de dos dimensiones que creamos en el primer ejercicio a la matriz identidad? ¿Por qué?
 - En caso de que no se puedan sumar, busca una solución para sumar una matriz identidad a otra. La solución puede ser crear un array nuevo.
 - Multiplicad la matriz identidad por la nueva matriz que os creasteis en el ejercicio anterior.
 - Dividid las dos matrices

Happy coding

NumPy II - Indexación

¿Cómo podemos indexar un array? ¿Qué métodos específicos tenemos para poder manipularlas?

NumPy II

La indexación de matrices se utiliza para acceder a elementos especificando sus índices dentro del array. Si tenemos un array llena de ceros y queremos poner un valor particular en un índice específico dentro del array, podemos usar el método de indexación de matrices.

¡Pero cuidado! Y aquí está la complejidad, la indexación de arrays funciona de manera muy diferente para arrays 1D (unidimensionales) , 2D (bidimensionales) o 3D (multidimensionales).



NumPy-II.ipynb 117KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso NumPy Indexación

- La Indexación en arrays es como en listas, siempre empieza en cero y siempre debe de ir entre corchetes.
- Indexación en función de las dimensiones
 - Unidimensional: `array[i]`, donde `[i]`=columna
 - Bidimensional: `array [i,j] = array[i][j]`, donde `[i]`=fila, `[j]`=columna
 - Multidimensional: `array[i,j,k] = array[i][j][k]`, donde `[i]`=array, `[j]`=fila, `[k]`=columna
- Indexación basado en rangos:
 - Unidimensional: `array[inicio:fin:salto]`
 - Bidimensional : `array[inicio:fin:salto,inicio:fin:salto] = array[inicio:fin:salto][inicio:fin:salto]`
 - Multidimensional: `array[inicio:fin:salto,inicio:fin:salto,inicio:fin:salto] = array[inicio:fin:salto][inicio:fin:salto][inicio:fin:salto]`
- Podemos filtrar aplicando operadores `<`, `>`, `>=`, `<=`, `==`.
 - `array[array (operador) condicion]`
 - Eg `array[array > condicion]`
 - Si queremos meter más de una condición: `&` o `|`
 - `array[array (operador) condicion1 anidamiento array (operador) condicion2]`
 - Eg `array[(array < 2) | (array > 5)]`
- Métodos:
 - `copy()` : hace una copia de nuestra array. Cualquier modificación que hagamos sobre la copia no afectará al array original.
 - `np.transpose()` : invierte los ejes de nuestra matriz o si se le especifica el orden, nos permuta los ejes de la matriz.
 - `np.swapaxes()` : intercambia dos ejes de la matriz, indicando sobre que ejes queremos realizar la operación
 - `np.reshape()` : cambia la forma de la matriz a la forma especificada.
- Para hacer operaciones entre matrices tienen que tener la misma forma y números de elementos!!!!

Pair Programming NumPy II

Seguimos trabajando con NumPy, en este caso practicaremos como acceder a distintos elementos de nuestros *arrays* y como cambiar su forma para poder hacer operaciones entre varios *arrays*.

1. Cread un *array* de dos dimensiones de 5 filas y 3 columnas. Sobre el *array* creado anteriormente, extraed:
 - El valor de la primera fila y la segunda columna.
 - Los valores de la segunda fila y las dos primeras columnas.
 - Extraed las filas pares (incluyendo la 0) y todas las columnas.
 - Extraed los valores que sean mayores que 0.5.
 - Extraed los valores menores que 0.2 o mayores que 0.5
2. Cread dos *array* de tres dimensiones:
 - El primero de 2 matrices, 5 filas y 3 columnas.
 - El segundo de 3 matrices, 2 filas y 5 columnas
 - Haced un análisis exploratorio de los *array* creados. Recordad que en el ejercicio de ayer nos creamos una función para esto.
 - Sumad los dos *arrays*. ¿Podéis? ¿Por qué?
 - Para cambiar la forma de un *array* y poder hacer operaciones entre *arrays* que tienen distintas formas hemos aprendido dos métodos:
 - `.reshape()`
 - `.transpose()`

Utiliza estas dos funciones para cambiar la forma de los *array* y hacer operaciones entre ellos.

Happy coding

Numpy III - Operaciones estadísticas y matemáticas

En este jupyter veremos que son las funciones universales y cuales son las principales que nos podemos encontrar dentro de la librería NumPy

NumPy III

Una **función universal** (abreviado `**_ufunc**`) es una función que realiza operaciones elemento a elemento de la matriz.

Las funciones universales en Numpy son funciones matemáticas simples. Numpy proporciona varias funciones universales que cubren una amplia variedad de operaciones.

Estas funciones incluyen funciones trigonométricas estándar, funciones para operaciones aritméticas, manejo de números complejos, funciones estadísticas, etc



Numpy-III.ipynb 122KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen NumPy 3: Operaciones matemáticas y estadísticas:

- Funciones Universales:
 - Son funciones matemáticas simples.
 - Operan sobre arrays de cualquier dimensión.
 - Deben de tener los arrays a los que apliquemos las operaciones la misma forma.
 - Algunas operaciones son:
 - `add()`
 - `multiply()`
 - `subtract()`
 - `divide()`
- Operaciones estadísticas:
 - Todas las operaciones se pueden hacer para la totalidad del array o por sus ejes. Los ejes son el contrario en pandas y en numpy:
 - Si `axis = 0` == columnas
 - Si `axis = 1` == filas
 - Si no se indica ejes, se calcula para todo el array.
 - `np.mean()` : Calcula la media de todos los elementos del array.
 - `np.std()` : Nos calcula la desviación estandar del array
 - `np.var()` : Nos calcula la varianza del array.
- Operaciones aritméticas:
 - `np.min()` Nos devuelve el mínimo
 - `np.max()` Nos devuelve el máximo
 - `np.sum()` Nos devuelve la suma del array
 - `np.cumsum()` : calcula la suma acumulada de todos los elementos del array
 - `np.cumprod()` : calcula la multiplicación acumulada de todos los elementos del array
 - `np.sqrt()` : la raíz cuadrada de los elementos positivos
 - `np.exp()` : calcula la exponencial
 - `np.mod()` : resto de la división
- Las operaciones de sumar, restar no son funciones universales simples
- Operaciones de comparación:
 - `np.any()` : comprueba que ALGÚN elemento cumpla la condición
 - `np.all()` : comprueba si TODOS los elementos cumplen la condición
 - Nos van a devolver valores booleanos (True/False)
- Operaciones trigonométricas:
 - `np.cos()` : nos calcula el coseno
 - `np.sin()` : nos calcula el seno
 -

- Otros métodos:
 - `np.tan()` : nos calcula la tangente
 - `np.sort()` : por defecto nos ORDENA por FILAS
 - `np.flip()` : nos da la vuelta (invierte) el orden de los elementos de nuestro array. Por defecto ordena TODO el array
 - `np.where()` : nos muestra los índices donde se cumple una condición dada.
 - `np.round()` : nos redondea al número de decimales que le especifiquemos.
- Los ejes son el contrario en pandas y en numpy:
 - El 0 son filas en pandas, columnas en numpy
 - El 1 son columnas en pandas, filas en numpy
 - Comentarios adicionales de los ejes:
 - Si Axis = 0, realizamos las operaciones sobre las filas, operativamente al aplicar operaciones utilizando este eje nos devuelve el mismo número de columnas que teníamos originalmente. Lo que ha realizado por dentro es colapsar los valores iterando por cada una de las filas
 - Si Axis = 1, realizamos las operaciones sobre las columnas, operativamente al aplicar operaciones utilizando este eje nos devuelve el mismo número de filas que teníamos originalmente. Lo que ha realizado por dentro es colapsar los valores iterando por cada una de las columnas
 - Si no se indica ejes, se calcula para todo el array.

Pair Programming NumPy III

En esta lección hemos aprendido algunas de las operaciones matemáticas más importantes que podemos aplicar a un *array*. Hoy pondremos en práctica algunas de estas operaciones, además de recordar algunas de los métodos aprendidos en otras lecciones y recordar como podemos iterar por un *array*.

- Cread un array de tres dimensiones de 2 matrices, 3 filas y 5 columnas cuyos valores solo tengan 1 decimales. Sobre el array creado anteriormente, extraed:
- Calculad la media del *array* y almacenarlos en una variable. Redondead a un decimal.
- Calculad el valor máximo del *array* y almacenarlos en una variable. Redondead a un decimal.
- Calculad el valor mínimo del *array* y almacenarlos en una variable. Redondead a un decimal.
- Cread un *array* vacío con la misma forma y dimensiones que el array creado anteriormente. Vamos a reemplazar los valores de la matriz vacía que nos acabamos de crear pero basándonos en los valores del primer array bajo las siguientes condiciones:
 - Si el valor del array es igual que la media, se reemplaza por "A".
 - Si el valor del array es mayor que la media, se reemplaza por "B".
 - Si el valor del array es menor que la media, se reemplaza por "C".
 - Si el valor es igual que el máximo, se reemplaza por "D".
 - Si el valor es igual que el mínimo, se reemplaza por "E".

```
# a modo de ejemplo. Si tenemos el siguiente array
[[[0.4, 0.7, 0.2, 0.9, 0.2],
  [0.1, 0.9, 0.6, 0.1, 0.2],
  [1., 0.2, 0.1, 0.8, 0.3]],

 [[0., 0.3, 0.4, 0.5, 0.1],
  [0.5, 0.7, 0.9, 0., 0.8],
  [0.5, 0.3, 0.5, 0.8, 0.1]]])

# sabiendo que la media es 0.4 y el máximo es 1.0 y el mínimo es 0.0

# nuestro array vacío debería quedar como este:
[[["A", "B", "C", "B", "C"],
  ["C", "B", "B", "C", "C"],
  ["D", "C", "C", "B", "C"]],

 [[["E", "C", "A", "B", "C"],
  ["B", "B", "B", "E", "B"],
  ["B", "C", "B", "B", "C"]]])
```

Pista Tendréis que iterar por el array y cambiar los valores de la matriz vacía. Para eso tendremos que usar distintos condicionales.

- Estupendo, tenemos un *array* con *strings*. Ahora realizaremos un ejercicio similar al anterior, pero en este caso nos crearemos un *array* vacío unidimensional de 30 elementos. Al final de este ejercicio acabaremos teniendo un *array* de tres dimensiones con la misma forma que el primero que nos creamos.

En este caso:

- Si el valor del *array* creado en el ejercicio anterior es "A" o "B" lo reemplazaremos por 1
- Si el valor del *array* creado en el ejercicio anterior es "C" o "D" lo reemplazaremos por 2
- Si el valor del *array* creado en el ejercicio anterior es "E" lo reemplazaremos por 3.

Una vez que hayas reemplazado todos los valores cambiad la forma del *array* para que sea de tres

dimensiones, con 2 matrices, 3 filas y 5 columnas.

Happy coding

NumPy IV - Conjuntos, distribuciones aleatorias

NumPy se utiliza mucho en teoría de conjuntos y por esto va a tener una serie de métodos asociados a este tipo de aproximaciones. Aprenderemos los principales métodos de teoría de conjuntos así como h

NumPy ofrece algunas funciones básicas de conjuntos para arrays unidimensionales. Entre las que veremos en esta lección incluimos:

- `np.unique()` : devuelve un array con los valores únicos de un array.
- `np.intersect1d()` : devuelve un array con los valores que se encuentran en ambos arrays.
- `np.union1d()` : devuelve un array con los valores que se encuentran en uno de los arrays.
- `np.in1d()` : devuelve un array con los valores que se encuentran en uno de los arrays.
- `np.setdiff1d()` : devuelve un array con los valores que se encuentran en uno de los arrays.
- `np.setxor1d()` : devuelve un array con los valores que se encuentran en uno de los arrays.

NumPy random permite generar arrays con distintos tipos de distribuciones. Entre las que veremos en esta lección incluimos:

- `np.seed()` : permite establecer una semilla para la generación de números aleatorios.
- `np.random.uniform()` : genera un array con valores aleatorios uniformes.
- `np.random.binomial()` : genera un array con valores aleatorios binomiales.
- `np.random.normal()` : genera un array con valores aleatorios normales.
- `np.random.permutation()` : genera un array con valores aleatorios permutados.

Por último, NumPy guarda los arrays en los ordenadores.



Numpy-IV.ipynb 137KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso NumPy IV

- Funciones de conjuntos:

- `np.unique()`: nos devuelve los valores únicos de un *arrays*
 - Argumentosopcionales:
 - `return_index` : Nos devuelve los indices del array de entrada
 - `return_inverse` : Nos devuelve los índices que nos permite reconstruir el array original(una vez aplicado el `flatten()`)
 - `return_counts` : Nos devuelve el número de veces que aparece repetido cada elemento único.
 - `np.intersect1d()`: los elementos comunes entre los dos *arrays*
 - `np.union1d()`: unión de los valores únicos ordenados entre dos *arrays*
 - `np.in1d()`: elementos de A que están en B. Nos lo devuelve como BOOLEANOS
 - `np.setdiff1d()`: nos devuelve los valores únicos ordenados de los que hay en A pero NO en B
 - `np.setxor1d()`: los valores únicos que están en A o en B pero NO en AMBOS
- Creación de arrays random que siguen ciertas distribuciones
 - `np.seed()`: plantamos una semillita y a partir de ahí todos los *arrays* que creemos tendrán los mismos valores.
 - `np.random.uniform()`:
 - `np.random.binomial()`: variable es "dicotómica", cara o cruz, si o no, enfermo o no... La probabilidad de que ocurra cada uno de esos eventos. Asumiendo que en inicio esa probabilidad es de 0.5 para cada uno de los eventos.
 - `np.random.normal()`: campana de Gauss.
 - `np.random.permutation()`: permutación aleatoria de una rango. Baraja las cartas.
- Guardar y Cargar:
 - `np.savetxt()`: guardar nuestro *array*. Lo haremos en formato .txt
 - `np.loadtxt()`: cargamos un *array* guardado anteriormente.

Pair Programming NumPy IV

Manos a la obra con la última lección de NumPy. En este caso pondremos en práctica como generar *arrays* que sigan ciertas distribuciones, así como las operaciones de teoría de conjuntos básicas.

1. ¿Cómo podemos crear dos *arrays* de dos dimensiones usando el método `random_sample` que tengan los mismos valores?
2. Cread los siguientes *arrays*:
 - Un *array* de tres dimensiones con distribución uniforme de $2 \times 3 \times 5$, con números entre 0 y 10, que no tengan decimales y extrae sus valores únicos almacenándolos en una variable.
 - Un *array* de tres dimensiones con distribución binomial de $2 \times 3 \times 5$, con probabilidad de 0.3 y número de pruebas igual a 10. Igual que antes, extrae sus valores únicos y guarda los datos en una variable.
3. En esta lección hemos aprendido algunos métodos de teorías de conjuntos. Usando los *arrays* de elementos únicos obtenidos en el ejercicio anterior realizad las siguientes operaciones:
 - Elementos comunes entre los dos *arrays*
 - Unión de los dos *arrays*
 - ¿Qué elementos están en el *array* derivado de la distribución uniforme y no en los únicos de la distribución binomial? Extrae los valores numéricos

⚠️ Explicar cada uno de los *outputs* que os salgan. Que significa cada una de estas operaciones que hemos ido extrayendo?
4. Cread dos *arrays* usando el método `random.randint` de Numpy con 80 elementos cada uno con números del 1-100, con los siguientes nombres `array1` y `array2`.
 - Extraed los valores únicos de cada uno de los *arrays* y los guardáis en variables en las que se les llame `array1` y `array2`.
 - Identificad los elementos presentes en el `array1` pero no en el `array2` y asignarlos a una variable que se llame `array3`.
 - Identificad los elementos presentes en el `array2` pero no en el `array1` y asignarlos a una variable que se llame `array4`.
 - Identificad los elementos presentes en ambos *arrays* y asignarlos a una variable que se llame `array5`.
 - ¿Cuál es la relación que existe entre todos estos *arrays*? **Pista** Tendréis que usar las longitudes de cada uno de los *arrays* creados.

Happy coding

Pandas

Datos

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de Pandas

En esta carpeta encontrareis los siguientes ficheros:

- **Sesión 1 Pandas I - Series y DataFrames**
 - `data.csv`
- **Sesión 2 Pandas II - Carga y Guardado de Datos**
 - `jobs.csv`
 - `aire.csv`
 - `espacios_protegidos.xlsx`
 - `residuos.json`
 - `airlines.sas7bdat` : este conjunto de datos lo encontraréis [aquí](#)
 - `10M.csv` : este conjunto de datos lo encontraréis [aquí](#)
 - `10M.pkl` : este conjunto de datos lo encontraréis [aquí](#)
 - `10M.parquet` : este conjunto de datos lo encontraréis [aquí](#)
- **Sesión 3 Pandas IV - Filtrado de Datos**
 - `Marketing-Customer-Analysis.csv`
- **Sesión 5 Pandas V - Merge Concat Join**
 - `survey2000.csv`
 - `survey2001.csv`
 - `survey2002.csv`
 - `species.csv`
- **Sesión 6 Pandas VI - Groupby**
 - `Marketing-Customer-Analysis.csv`
- **Sesión 7 Pandas VII - Apply**
 - `Marketing-Customer-Analysis.csv`



`data.csv` 194B
Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



jobs.csv 34MB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



aire.csv 20KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



espacios_protegidos.xlsx 22KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



residuos.json 24KB

Code

Descarga este fichero para poder hacer los ejercicios que os planteamos.



Marketing-Customer-Analysis.csv 1MB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



survey2000.csv 59KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



survey2001.csv 61KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



survey2002.csv 85KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



species.csv 2KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



titanic.csv 104KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.

Pandas I- Dataframes - Series

¿Qué es Pandas? ¿Qué es una Serie? ¿Cómo podemos crear una Serie? ¿Para qué las podemos usar? ¿Qué es un dataframe? ¿Para qué lo vamos a usar? ¿Cómo podemos crear dataframes en Pandas?

Pandas I - Series y DataFrames

Una Serie es la estructura más simple que proporciona Pandas, y se asemeja a una columna en una hoja de cálculo de Excel. Son estructuras unidimensionales que contienen un array de datos (de cualquier tipo soportado por NumPy). Veremos como crearlas, y cuáles son sus principales características.

Un DataFrame es una estructura de datos con dos dimensiones en la cual podremos guardar datos de distintos tipos (strings, enteros, decimales, factores, etc.). Son como nuestros conocidos Excel, solo que nos van a permitir realizar muchas más operaciones y de una forma más sencilla. En esta lección aprenderemos como crearlos y cuáles son sus propiedades básicas.



Pandas-I-Series.ipynb 44KB
Binary

Descarga este Jupyter y ábrelo en VS Code.



Pandas-I-DataFrames.ipynb 88KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Intro Pandas

- Pandas: Es una librería de python que sirve para trabajar con conjuntos de datos (similar a excel) utilizando tablas.
- Tiene 3 estructuras de datos diferentes:
 - Series: Una dimensión.
 - Dataframes: Dos dimensiones.
 - Panel: Tres dimensiones.
- Series:
 - Estructura más simple compuesta por un array de datos, sería equivalente a una columna de excel.
 - Para crear series podemos:
 - Podemos crear una serie vacía con `pd.Series()`
 - Podemos crearlo a partir de una lista: `pd.Series(lista)`
 - Podemos crearlo a partir de un array: `pd.Series(array)`
 - Podemos crearlo a partir de un diccionario, utilizando las keys o los values, si no especificamos, nos tomará como índices las claves del diccionario y como valores, los valores del diccionario:
 - `pd.Series(diccionario)`
 - `pd.Series(diccionario.keys())`
 - `pd.Series(diccionario.values())`
 - Podemos crearlo utilizando un escalar: - `pd.Series(valor_numerico)`
 - Propiedades de las series:
 - `.index` : Te devuelve los índices de la serie
 - `.values` : Te devuelve los valores de la serie
 - `.shape` : Nos devuelve la forma de la serie
 - `.size` : Nos devuelve el número de elementos de la serie.
 - `.dtypes` : Nos devuelve el tipo de datos de nuestra serie.
 - Para indexar una serie debemos llamarlo utilizando corchetes y el valor del índice. Si ponemos más de un valor en forma de lista nos devolverá una los elementos indicados.
 - `serie[valor]`
 - `serie[valor1, valor 2]` : Nos devuelve dos elementos, basado en los índices especificados
 - `serie[valor1 : valor2]` : Nos devuelve los valores en el rango entre ambos
 - `serie[etiqueta]` : Cuando quieras llamar al elemento por la etiqueta que tiene asignada
 - `drop` : Eliminamos filas de nuestra serie
 - Podemos hacer operaciones con las series, siempre que contentan los mismos índices, si no, nos devolverá NaN:
 - Suma: `+` o `np.add()`
 - Resta: `-` o `np.sub()`

- Multiplicacion: `*` o `np.mul()`
- Division: `/` o `np.div()`
- Módulo: `mod()`
- Exponencial: `pow()`
- Con `fill_value`: nos autocompletará los NaN con los valores de nuestra serie, sumándose al valor del elemento original
- Filtrado booleano:
 - Utilizando los operadores de comparación `<`, `>`, `>=`, `<=`, `==`.
 - Devuelve True si se cumple la condición, False si no se cumple.
- Búsqueda de Nan:
 - `isnull()` : Devuelve los valores que faltan en la serie, True si es NaN, False si no.
 - `isnotnull()` : Devuelve aquellos valores que no son NaN de la serie, True si no es nulo, False si es NaN.
- DataFrames:
 - Para datos tabulares.
 - Creación de dataFrames:
 - `pd.DataFrame(data, index, column)`
 - Podemos pasar listas, diccionarios y arrays para crear DataFrames
 - Selección de valores de un DataFrame:
 - `df.loc` : Para seleccionar los elementos en base a las etiquetas de la filas o columnas.
 - `df.loc[:]` es equivalente a `df`
 - `df.iloc` : Para seleccionar basado en los valores de los índices
 - Basándonos en condiciones con operadores comparativos:
 - Operadores comparativos como `>`, `<`, `>=`, `<=`, `==` o `!=`
 - Multiples condiciones: Animando los filtros con `&` o `|`
 - Utilizando `.loc`
 - `df.loc[df.columna > valor]`
 - Utilizando `.iloc()` deberemos pasar el filtro como list()
 - `df.loc[list(df.columna > valor)]`
 - Creación de columnas:
 - `df[columna_nueva] = valores`
 - `df.assign(nombre_columna_nueva=valores)`
 - `df.insert(indice_posicion, nombre_columna, valores)`

Pair Programming Pandas I

Empezamos los ejercicios de *pair programming* con Pandas. En esta lección hemos aprendido que son las Series y DataFrames y como podemos crearlos en Python.

El objetivo de la clase de hoy será crear unos cuántos *DataFrames* y *Series* a través de los siguientes ejercicios:

1. ¿Qué diferencia hay entre una *Serie* y un *DataFrame*?
2. Momento de trabajar con Series. Contestad a las siguientes preguntas:
 - Crea dos listas:
 - Una que se llame "valores" que tenga 7 números aleatorios.
 - Otra que se llame "índices" que tenga 7 *strings* que se llamen como los días de la semana.
 - A partir de estas dos listas cread una *Serie* donde la variable "valores" sean los valores de la *Serie* y la variable *índices* sean los índices de la *Serie*.
 - A partir de las dos listas anteriores cread un diccionario usando un bucle *for*. Para hacer esto tenemos el método `zip` que nos permite iterar por dos listas a la vez. Más información sobre este método [aquí](#).
 - Crea una *Serie* con el diccionario creado en el punto anterior.
 - ¿Qué forma tiene esta última *Serie*?
 - ¿Qué tipo de datos tiene?
 - ¿Qué tamaño tiene?
 - ¿Cuáles son los índices de la *Serie*?
 - ¿Y los *values*?
 - Extraed el valor para el índice "lunes"
 - Extraed el valor para el índice "lunes", "martes" y "miércoles".
 - Extraed el valor para el índice "lunes" y "domingo".

3. Es el turno de los *DataFrames*

- Cread un *array* de dos dimensiones con 4 filas y 3 columnas.
- Convertid el *array* en un *DataFrame* donde las columnas se llamen "España", "Francia" e "Italia".
- Descargad el archivo "medallas" y cargad el *DataFrame* con la siguiente línea de código:

```
df = pd.read_csv("medallas.csv", index_col=["País"])
```

NOTA Para evitar problemas guardar el archivo `medallas.csv` a la misma altura (en la misma carpeta) que el jupyter donde estamos trabajando.

- Usando el `loc` extraed todas las columnas de la fila de Tailandia.
- Usando el `loc` extraed todas las columnas de China e Irán.
- Usando el `loc` extraed solo las columnas de "Oro", "Plata" y "Bronce" de China e Irán.
- Usando el `iloc` extraed la información de la India. Devolved todas las columnas.
- Usando el `iloc` extraed todas las columnas de los países que estén en posición impar. La primera que nos tendría que salir es Japón y la última Corea del Norte.
- Utilizando el código del ejercicio anterior, seleccionad únicamente las columnas "Oro", "Plata" y "Bronce".
- Contestad a las siguientes preguntas:
 - ¿Qué país consiguió más de 200 medallas de oro?
 - ¿Qué país consiguió más de 200 medallas de oro y más de 1000 de plata?

- ¿Qué país consiguió más de 200 medallas de oro o más de 700 de bronce?
- Como vemos hay una columna que se llama `Unnamed: 0` que no nos interesa. Eliminad esa columna.
- Cread una nueva columna que se llame "total" que sea el resultado de la suma de todas las medallas obtenidas por cada país.



medallas.csv 267B
Binary

Descarga este csv para poder hacer los ejercicios de *pair programming*.

Happy coding

Pandas II - Carga y guardado de datos

En este jupyter aprenderemos como cargar y guardar datos en Pandas.

Pandas II - Carga y Guardado de Datos

Pandas es una herramienta muy potente que nos va a permitir trabajar con distintos tipos de ficheros. Entre ellos podemos encontrar:

- csv
- txt
- excel
- parquet
- sas
- spss
- json

En esta lección aprenderemos como cargar todo este tipo de ficheros y como guardarlos una vez que lo hayamos manipulado.



Pandas-II-Carga-Guardado-Datos.ipynb 53KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso pandas II - Lectura y guardado

- Cargar datos: De forma general para leer datos utilizaremos:
`pd.read_extension(nombre_fichero.extension)`
- Guardar datos: De forma general para guardar datos utilizaremos:
`nombre_dataset.to_extension(nombre_fichero.extension)`
- Tipos de ficheros:
 - csv (comma separated values): Se consideran datos puros y son legibles en un editor de texto.
 - Para leerlos: `pd.read_csv()`
 - Nos permite incluir otros separadores: ; , | , ... Y será necesario especificarlo, ya que si no lo hacemos leera el fichero de forma incorrecta.
 - Para guardar ficheros con esta extension:
`nombre_dataset.to_csv(nombre_fichero.csv)`
 - excel: Fichero de tipo excel.
 - Para leerlos: `pd.read_xls()` o `pd.read_xlsx()` Dependiente de la version en la que se guardó el archivo. Y guarda las propiedades del propio excel. -- Para guardar ficheros con esta extension: `nombre_dataset.to_excel(nombre_fichero.xls / nombre_fichero.xlsx)`
 - json (JavaScript Object Notation): Guarda los datos de forma estructurada y como diccionarios, muy extendido en BBDD noSQL e Internet.
 - Para leerlos: `pd.read_json()` Y por lo general habrá que hacer transformaciones adicionales para poder tenerlo como dataframe.
 - Para guardar ficheros con esta extension:
`nombre_dataset.to_json(nombre_fichero.json)`
 - Para crear DataFrames desde el portapapeles:
 - Para hacerlo `pd.read_clipboard()`, por defecto tiene como separador \s+ .
 - pickle: Un formato único de Python, que nos sereliaza los datos(los guarda como diccionarios) y preserva los tipos de datos, tal y como estan en el dataframe original.
 - Para leerlo: `pd.read_pickle()`
 - Para guardar ficheros con esta extension:
`nombre_dataset.to_pickle(nombre_fichero.pkl)`
 - parquet: Es un formato de datos serializado (diccionarios)
 - Para leerlo: `pd.read_parquet()`
 - Para guardar ficheros con esta extension:
`nombre_dataset.to_parquet(nombre_fichero.parquet)`
 - SAS: Es un archivo de SAS, es un programa en especifico(como lo es excel).
 - Para leerlo: `pd.read_sas()`
 - Para guardar ficheros con esta extension:
`nombre_dataset.to_sas(nombre_fichero.extension)`
 - SPSS: Es un archivo de SPSS, es un programa en especifico SPSS (como lo es excel).
 -

- Para leer: `pd.read_spss()`
- Para guardar ficheros con esta extensión:
`nombre_dataset.to_csv(nombre_fichero.csv)`

CSV	PICKLE	PARQUET
✓ lectura humana	✓ guardado/carga rápida	✓ guardado/carga rápida
✓ en todas las plataformas	✓ menos espacio en disco	✓ menos espacio en disco que pickle
más lento	no es legible para las personas	✓ soportado por muchas plataformas
más espacio en disco	sólo para python	es menos legible para los humanos
no conserva los tipos en algunos casos		

Pair Programming Pandas II

En este ejercicio de *pair programming* pondremos en práctica como cargar distintos tipos de archivos en Python usando Pandas.

Al final de la explicación encontrareís una serie de ficheros con distintos formatos.

Los objetivos de este *pair programming*:

- Id cargando todos los ficheros en vuestro jupyter.
 - Si os da error leer el *pickle* será por la versión de pandas. Podéis guardar el csv como *pickle* y cargarlo de ahí.
- Para poner el práctica el `read_clipboard` elegir alguna tabla que os podáis encontrar en google y usad dicho método para traer los datos al jupyter notebook. Una vez que los hayáis cargado, guardad esos datos en los siguientes formatos:
 - csv
 - excel
 - json
 - pickle



quejas.csv 793KB

Binary

Descarga este csv para poder hacer los ejercicios de *pair programming*.



quejas.json 2MB

Code

Descarga este json para poder hacer los ejercicios de *pair programming*.



quejas2.xlsx 15KB

Binary

Descarga este xlsx para poder hacer los ejercicios de *pair programming*.



algas.pickle 724KB

Binary

Descarga este pickle para poder hacer los ejercicios de *pair programming*.

Happy coding

Pandas III - Métodos Pandas

En esta lección aprenderemos los principales métodos de Pandas. ¿Cómo podemos explorar nuestro dataframe? ¿Podemos modificar algunas de sus características?

Pandas III - Métodos de Pandas

¿Qué es la exploración de un *dataset*? Conoceremos cuantas filas y columnas tenemos en nuestro *DataFrame*, cuantos valores nulos, los nombres de nuestras columnas, sacar los principales estadísticos (media, mediana, desviación estándar etc.).

Además aprenderemos como reemplazar valores dentro de nuestro set de datos e incluso eliminar! O como podemos hacer re-asignación de columnas en función de otras columnas.



Pandas-III-Metodos-Pandas.ipynb 82KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen pandas III - Métodos Pandas

- **Métodos de pandas para explorar DataFrames.**

- Obtener muestras del DataFrame:

- `df.head(n_filas)` : Te devuelve las primeras 5 filas por defecto. Si introducimos un número te devuelve el número especificado de filas.
- `df.tail(n_filas)` : Te devuelve las ultimas 5 filas.Si introducimos un número te devuelve el número especificado de filas.
- `df.sample(n_filas)` : Te devuelve una fila aleatoria del DataFrame.Si introducimos un número te devuelve el número especificado de filas.

- Atributos del DataFrame:

- `.columns` : Devuelve los nombres de las columnas
- `.shape` : Devuelve el numero de filas y columnas
- `.dtype` : Nos devuelve los tipos de datos de cada columna.

- Métodos:

- `.describe()` : Resumen estadístico de las columnas numéricas
- `.info()` : Resumen de los tipos de datos de cada columna, nombres, numero de valores faltantes, etc.
- `.unique()` : Nos devuelve los valores únicos de una columna, es necesario especificar la columna. Sin dar información de si los valores están repetidos o no en la columna.
- `.value_counts()` : Nos devuelve una cuenta de las veces que aparece un valor en nuestra columna. Se puede utilizar para identificar los valores repetidos.

- Para identificar valores faltantes:

- Ambos métodos nos devuelven los mismos resultados:
 - `.isna()`
 - `.isnull()`
- Se puede aplicar funciones para obtener el número de valores faltantes.

- Correlación:

- `.corr()` : Devuelve la correlación por pares de columnas, siendo esta correlación la relación de dependencia entre estas dos variables.

- Métodos de manipulación del dataframe:

- Convertir una columna a índice:

- `.set_index(['nombre_columna'], inplace = False)`:
 - Si `inplace = False`, te añade una nueva columna y lo imprime por pantalla, pero no modifica el dataframe
 - si `inplace = True`, te modifica el dataframe.

- `.drop()` : para eliminar columnas/filas

- `.rename()` : para cambiar columnas/filas de nombre

- `.cut()` : separar los elementos en intervalos, según los bins que le damos

- `.replace()`: para sobreescribir datos a través de su valor original. NO a través de su posición

en el df

Pair Programming Pandas III

De aquí en adelante, vamos a trabajar con el mismo dataset. Se trata de un conjunto de datos que contiene información sobre ataques de tiburones en distintas partes del mundo. ¿El problema de los datos? Están terriblemente sucios!!! En las próximas lecciones iremos limpiando este `csv` y sacaremos conclusiones usando la visualización.

Tareas a realizar en este *pair programming*

1. Cargar el dataset

Pista Si hacemos la carga del dataset sin el parámetro `encoding` nos puede dar error (puede que no os de error, si es así menos problemas!). [Aquí](#) tenéis una lista de los encodings disponibles en pandas. Buscad por los `encoding` de tipo ISO.

2. Haced una exploración básica del *DataFrame*:

- ¿Cuántas filas y columnas tiene el *DataFrame*?
- Mostrad las primeras cinco filas y las últimas cinco filas del *DataFrame*.
- Mostrad 2 filas al azar del *DataFrame*.
- ¿Cuál es el nombre de las columnas del *DataFrame*?
- ¿Cuáles son los principales estadísticos del *DataFrame*? Mostrad por separado las variables categóricas y las variables numéricas.
- ¿Cuál es el porcentaje de valores nulos en cada columna?
- Sacad los valores únicos y su frecuencia de las variables numéricas. Almacenad los resultados en un diccionario. Uno para cada variable.
- Cambiad el nombre de las columnas para que todas estén en minúsculas y reemplazad los espacios por `_`.
- Eliminad las columnas de `Unnamed: 22` y `Unnamed: 23`, `Case Number.1`, `Case Number.2`, `original order`, `Investigator or Source`, `pdf`, `href formula`.
- Establece la columna `Case Number` como índice del *DataFrame*.

3. Guarda el `csv` que no tiene esas columnas para seguir usándolo mañana.



attacks.csv 2MB

Binary

Descarga este csv para poder hacer los ejercicios de *pair programming*.

Happy coding

Pandas IV - Filtrado de datos

En este jupyter veremos como podemos filtrar los datos de nuestro dataset en función de condiciones específicas.

Pandas IV

Ya sabemos que pandas es potente, y que presenta muchas herramientas que nos pueden ayudar a conocer los datos que tenemos. Una de las características de Pandas es que nos va a permitir filtrar en función de:

- Valores de una columna
- Valores de múltiples columnas
- Índices y condiciones, donde volveremos a recordar los `loc` e `iloc`
- Métodos específicos de Pandas:
 - `isna()`
 - `str.contains()`
 - `notnull()`
- Métodos de NumPy
 - `np.where()`
 - `np.select()`



Pandas-IV-Filtrado-datos.ipynb 169KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen Pandas IV: Filtrado de datos

- Formas de filtrar datos

- Filtrar por condición

- `df[“Column”] == “Nevada”` : una serie de True y False
 - `df[df[“Column”] == “Nevada”]` : filtro mi DataFrame por las filas que sean True

- Combinar condiciones con & (and), | (or), ~ (not)

- `(df[“Provincia”] == “Madrid”) | (df[“Provincia”] == “Barcelona”)` : Una Serie de True y False
 - `df[(df[“Provincia”] == “Madrid”) | (df[“Provincia”] == “Barcelona”)]` : un DataFrame filtrado
 - `df[~((df[“Provincia”] == “Madrid”) | (df[“Provincia”] == “Barcelona”))]` : filtrado al revés

- Podemos guardar en variables los filtros:

- `filtro = (df[“Provincia”] == “Madrid”) | (df[“Provincia”] == “Barcelona”)`
 - `filtro_inverso = df[~filtro]`
 - `df[“Column”].isin(values)`
 - `df[“Provincia”].isin([“Madrid”, “Barcelona”])`
 - `str.contains()` : filtrar con un patron de regex
 - `notnull()` : filtrar por lo que no sea nulo
 - Reemplazar valores en el DataFrame
 - `np.where()` : para reemplazar elementos que cumplan una condición
 - `np.select()` : requiere una lista de condiciones y una lista que se devuelva.

- Guardar el dataset filtrado

Tenemos que hacer la asignación de variable a un nuevo DataFrame

- `df_nevada = df2[df2[“Column”] == “Nevada”]`

Pair Programming Pandas IV

Hoy seguiremos trabajando con el csv de ataques que exploramos ayer. Para eso, ayer guardasteis al final del *pair programming* al que le habíamos eliminado algunas columnas. Utilizad ese fichero.

En la lección de hoy tendréis que ir contestando una serie de preguntas que os planteamos. Para ello tendréis que aplicar los conocimientos adquiridos en la clase invertida sobre como filtrar datos en el *DataFrame* y como reemplazar valores en el *DataFrame*.

1. Cargad el csv que guardasteis ayer.

Antes de seguir con las preguntas, este DataFrame está terriblemente sucio. En las próximas sesiones lo iremos limpiando. Para evitar posibles errores con los nulos vamos a eliminar todos los nulos del *DataFrame*.

Para ello vamos a utilizar el método `.dropna()` de pandas. Tendréis que ejecutar la siguiente línea de código para eliminar los nulos del *DataFrame* y poder seguir haciendo el resto de los ejercicios.

```
df.dropna(inplace = True)
```

¿Cuántas filas se han eliminado con esta línea de código?

2. ¿Cuál es el número de ataques que se han producido en Estados Unidos?

3. ¿Y en España?

- 3.1 ¿En qué regiones se han producido los ataques (columna `location`) ?
- 3.2 ¿Cuántos ataques se han producido en cada área (columna `area`) ?

4. Extraed un *DataFrame* nuevo con los ataques que se han producido en Australia y Estados Unidos.

Nota Este ejercicio se puede resolver usando dos condiciones de filtrado o usando el método `isin()`. Hacedlo con ambas herramientas.

- 4.1 ¿Cuántos ataques se han producido en cada país?
- 4.2 ¿Ha habido ataques en Australia a gente que estuviera remando (lo encontrareis como *Paddling* en el *DataFrame* en la columna `activity`)?
- 4.3 ¿Cuántos ataques se han producido en Australia antes del 2000?
- 4.4 ¿Y después del 2000?

5. Extraed únicamente los ataques a mujeres.

6. ¿Cuántas mujeres fueron atacadas en España?

7. Extraed los ataques que se hayan producido en el año 2000, 1997 y 2006. Utilizad el método `isin()` para resolver este ejercicio.

8. Si extraemos los valores únicos de la columna `species` nos podemos dar cuenta que está un poco sucia y extraer información útil de la misma se puede convertir en algo difícil. En este ejercicio tendréis que extraer los ataques que han ocurrido por tiburón blanco (lo encontraremos en la columna `species` como algo que contenga `white` o `White`). Usad el método `str.contains()` para resolver este ejercicio.

9. Extraed ahora los ataques por tiburón blanco o tigre. Utilizad el método `str.contains()` para resolver este ejercicio.

10. Utilizando la columna de `year` cread una nueva columna llamada `siglo` que tiene `siglo-XX` o `siglo-XXI`. Donde el año sea menor que 2000 pondremos `siglo-XX` y el resto de los años `siglo-XXI`.

11. Guardad como un archivo csv nuevo. Lo usaremos en el *pair programming* de mañana.

Happy coding

Pandas V - Unión de datos

En este jupyter veremos como podemos unir varios DataFrames que comparten información en común, ya sea solo algunas columnas o todas.

Como ya hemos visto, las Series y DataFrames de Pandas son potentes herramientas para explorar y analizar datos. Parte de su poder proviene de un enfoque multifacético para combinar conjuntos de datos separados. Con Pandas, puedes fusionar, unir y concatenar tus conjuntos de datos, permitiéndote unificar y comprender mejor tus datos mientras los analizas.

En esta lección, aprenderemos cómo y cuándo combinar sus datos en Pandas con:

- `concat()` : para combinar DataFrames a través de filas o columnas.
- `merge()` : para combinar datos en columnas o índices comunes.
- `join()` : para combinar datos en una columna clave o un índice.

 Pandas-V-Merge-Concat-Join.ipynb 65KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Pandas V - Unión de datos

- Para unir todo en un único `csv` podemos usar
 - `pd.concat()`: es el más fácil, los nombres de las columnas se tienen que llamar igual. Los ejes:
 - `axis = 0` filas DEBAJO
 - `axis = 1` filas a la DERECHA
 - `pd.merge()`: es el más común para añadir nuevas columnas. Podemos incluir:
 - `how`: especificamos como unimos los dataframes, puede ser `inner`, `right`, `left`...
 - `on` y `right_on/left_on`:
 - `on`: si los nombres de las columnas por las que queremos unir son iguales
 - `right_on/left_on`: si los nombres de las columnas por las que queremos unir son distintas
 - `pd.join()`: es como un merge, pero UNA DE LAS COLUMNAS POR LAS QUE QUEREMOS UNIR TIENE QUE SER UN ÍNDICE.
 - `rsuffix` y `lsuffix`: cuando tenemos columnas que se llaman igual, pero no son las columnas que usaremos para unir los dataframes, podemos usar estos parámetros para especificar el nombre "nuevo".
 - `.reset_index()`: cuando unimos tablas que tienen los mismos índices es aconsejable hacer un `reset_index()` para que nuestros vayan del 0-n sin que se repita ninguno.

Pair Programming Pandas V

En este ejercicio pondremos en práctica los conceptos básicos para juntar varios `csv` que comparten información.

Trabajaremos con tres ficheros diferentes:

- El `csv` que creamos en el ejercicio de *pair programming* Pandas IV, dónde recordamos eliminamos ciertas columnas que no eran necesarias, y habíamos creado nuevas.
- El `csv` llamado `attacks_historico` donde tenemos información sobre los ataques de tiburón previos al año 1800, este fichero lo tenéis un poco mas abajo para descargar.
- El `csv` llamado `attacks_adicional` donde tenemos información sobre los ataques de tiburón. de nuevo, este fichero lo tenéis abajo para descargar.

El principal objetivo de este ejercicio de *pair programming* es acabar teniendo un único *DataFrame* con toda la información contenida en los tres `csv` anteriores.

Para llevar a cabo nuestro objetivo tendréis que seguir los siguientes pasos:

1. Cargad los tres ficheros nombrados anteriormente y los exploramos para familiarizarnos con ellos y saber que información tienen en común y debatir con tu compañera como podríamos juntar toda la información.
2. Chequead que los nombres de las columnas ver si se llaman iguales entre *dataframes*. ¿Están todas en minúsculas? ¿Tienen espacios? En caso de que no sean todas iguales unificad el nombre de las columnas.
3. En este momento nos habremos dado cuenta que el *DataFrame* que generamos en el *pair programming* Pandas IV es similar al *DataFrame* de los datos históricos. Si recordamos, eliminamos algunas columnas en la sesión III que no eran útiles, pero esas columnas están todavía en el histórico. Si no fuera por eso podríamos unir esos *dataframes*. Realizad los cambios que creáis necesarios para que los *dataframes* sean iguales y luego juntadlos usando el mejor método.
4. Es el momento de unir la información de todos los ataques de tiburón que acabamos de crear con el *DataFrame* de información adicional. Debatid que método es el más correcto para unir toda esta información y ejecutadlo.
NOTA Cuando hagáis la unión considerad que la información que más nos interesa es la del *DataFrame* que creamos en el apartado anterior, lo cual determinará el tipo de unión.
5. Guardad el dataframe nuevo en un `csv` que usaremos mañana en la clase de *pair programming*

Archivos



attacks_historico.csv 56KB

Binary

Descarga este csv para poder hacer los ejercicios.



attacks_adicional.csv 393KB

Binary

Descarga este csv para poder hacer los ejercicios.

Happy coding

Pandas VI - Groupby

En este jupyter veremos como podemos unir varios dataframes que comparten información en común, ya sea solo algunas columnas o todas.

En el análisis exploratorio de datos, a menudo queremos analizar los datos por algunas categorías. En SQL, la sentencia GROUP BY agrupa las filas que tienen los mismos valores. En Pandas, la operación GROUP BY de SQL se realiza mediante el método `groupby()`.



Pandas-VI-Groupby-Pandas.ipynb 106KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Pandas VI - Groupby

- Usos:

- Analiza datos de algunas categorías
- Divide, aplica (un estadístico) y agrupa (en función de la columna o columnas que especifiquemos)
- Si no le pasamos un estadístico nos devuelve un objeto de tipo groupby
- Filtra los datos

- Para convertir el resultado del groupby a un DataFrame:

- EL_RESULTADO_DEL_GROUPBY.reset_index()
- pd.DataFrame(EL_RESULTADO_DEL_GROUPBY)

- Estadísticos:

- .count()
- .describe()
- .sum()
- .mean()
- .median()
- .min()
- .max()
- .var()
- .std()
- .size()

- Se puede sacar un estadístico para todas las columnas o para una sola columna:

```
# PARA TODAS LAS COLUMNAS
df.groupby("gender").count()
```

```
# PARA UNA COLUMNA
df.groupby("gender")["num_polices"].count()
```

- También podemos incluir múltiples condiciones

```
# ESTAMOS HACIENDO UNA AGRUPACIÓN POR GÉNERO Y TAMAÑO DE VEHÍCULO
df.groupby(["gender", "vehicule_size"])["num_polices"].count()
```

- Si quiero calcular más de un estadístico == .agg(TIENE QUE IR EN FORMATO LISTA)

```
df.groupby("gender")["col1"].agg(["mean", "std"])
```

- Nulos:

- Tenemos el parámetro `dropna`.
- El groupby por defecto **LOS IGNORA**, es decir, `dropna = True`.
- En caso de que queramos que **nos incluya** los nulos tendremos que establecer `dropna = False`.

Pair Programming Pandas VI

Es el momento de poner en práctica los `groupby` de pandas. Para esto vamos a trabajar con el `csv` creado en el ejercicio de *pair programming* anterior.

El objetivo de este ejercicio es agrupar los datos para contestar las siguientes preguntas:

1. ¿Cuántos ataques hubo por país? Devuelve un *DataFrame*.
NOTA Ordena los resultados de mayor a menor. Esto todavía no lo hemos visto, pero un *DataFrame* lo podemos ordenar con `.sort_values()`. [Aquí](#) tenéis una explicación de cómo hacerlo.
2. ¿Cuántos ataques ha habido por año? Devuelve un *DataFrame*. De nuevo ordena los resultados de mayor a menor.
3. ¿Cuántos ataques hubo por año y sexo? Devuelve un *DataFrame* solo del año 2000 en adelante.
4. Primero extraed solo las filas que correspondan a USA, AUSTRALIA y SOUTH AFRICA. ¿Cuántos de los ataques fueron letales por sexo? Devuelve un *DataFrame*.
5. Usando el *DataFrame* creado en el ejercicio anterior. ¿En qué país hubo más ataques?
6. ¿Qué y cuántos tipos de ataques hubo por país?

Happy coding

Pandas VII - Apply

En este jupyter aprenderemos que es un `apply` y cómo y cuándo lo podemos usar.

Pandas VII - Apply

El método `apply` usa una función (tanto definida por nosotras como las propias de Python) a cada elemento a lo largo de la fila o columna de nuestro *DataFrame*.

También conoceremos el método `ExcelWriter` que nos permite escribir un *DataFrame* en un archivo de Excel. Además de ver como podremos guardar distintos datos resultado de varias operaciones como un `groupby`, un `apply` etc en un único excel (cada resultado en una hoja diferente).



Pandas-VII-Apply-Pandas.ipynb 81KB

Binary

[Descarga este Jupyter y ábrelo en VS Code.](#)

Resumen Pandas VII - Apply

- Utilizamos el metodo `.apply()` para aplicar una misma función a las filas o columnas de un objeto *DataFrame* de Pandas.
- Sintaxis general: `df['nombre_columna'].apply(funcion_a_aplicar)`
- Utilizamos el metodo `apply` con lambdas, cuando queremos aplicar la misma función a varias columnas o utilizar funciones con más de un argumento.
 - Sintaxis general: `df.apply(lambda x:funcion_a_aplicar(argumento1,argumento2))`
 - Si no tenemos una función definida previamente podemos aplicarlo directamente. Eg:
`df.apply(lambda x:df[columna_1]/df[columna_2])`
- `ExcelWriter`: Nos permite escribir diferentes conjuntos de datos sobre diferentes hojas de excel y escritura secuencial de diferentes datasets tras aplicar transformaciones a diferentes conjuntos de datos

```
with pd.ExcelWriter("nombre_fichero.xlsx") as writer: # donde queremos guardar el excel  
    df1.to_excel(writer, sheet_name="nombre_hoja1") # el dataframe que queremos guardar y como  
    df_copia.to_excel(writer, sheet_name="nombre_hoja2") # el dataframe y como queremos que sea
```

Pair Programming Pandas VII

En el ejercicio de ayer estuvimos viendo el `groupby` y pudimos observar que los datos estaban un poco sucios... . En los ejercicios de hoy pondremos en práctica los `apply` , y nos crearemos algunas funciones que nos van a permitir limpiar un poco los datos, para que podamos extraer conclusiones de una forma más sencilla.

Manos a la obra ! Los objetivos de la clase de hoy son:

1. Cread una columna nueva y una función que nos de el mes en el que ocurrió el ataque. Tened en cuenta que no todas las filas tienen la misma estructura y que puede que no haya la información de mes. En ese caso devolved un nulo (NaN).
Pista Podéis usar regex.
2. Vamos con la columna `fatal_(y/n)` . Lo primero que tenéis que hacer es evaluar los valores únicos que hay. Esperaríamos tener solo dos valores: `y` y `n` , pero tristemente no será así . Cread una columna nueva y una función que devuelva únicamente `y` o `n` y que devuelva un nulo (NaN) si no se encuentra el valor.
3. Seguid la misma lógica que en el ejercicio anterior pero con la columna `sex` . Pero en este caso vamos a incluir un retito más: En este caso querremos tener dos valores únicos `F` y `M` . De nuevo en la columna hay otros caracteres raros como `'M'` , `'lli'` , `'N'` , `'..'` . En esos casos, queremos que se reemplacen por `M` o `F` de forma aleatoria (tendréis que usar la librería `random`). En caso de que sea nulo, que la función devuelva nulo (`np.nan`).
4. Una vez que hayáis terminado eliminad las columnas originales conservando las nuevas.
5. Guardad el `dataframe` en un csv que usaremos más adelante

Happy coding

Visualización

Visualización I - Matplotlib

¿Qué es la visualización? Entenderemos la importancia de realizar correctas visualizaciones para entender los datos. Y a realizar gráficas haciendo uso del paquete Matplotlib

Visualización I

La visualización de datos consiste en la realización de gráficas a partir de un conjunto de datos. Generalmente las gráficas se utilizan para conocer en detalle los datos con los que se han de trabajar y así mismo sirve para poder extraer la información contenida de forma visual para poder explicar o dar respuesta a preguntas sobre los mismos.

Como se suele decir: "Una imagen vale más que mil palabras."

[Matplotlib](#) es una librería de Python que se utiliza para crear gráficos de todo tipo en Python. Ya que nos permite realizar gráficas estáticas, animadas e interactivas, con funciones como zoom, actualización, etc. Veremos como crear gráficas utilizando este potente paquete de Python!



Viz1.ipynb 278KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



Iris.csv 5KB

Binary

Descarga este csv para poder desarrollar la lección.

Repaso Visualización I - Matplotlib

Con `figure`, creamos una nueva figura de Matplotlib. Las diferentes partes de una gráfica de Matplotlib son las siguientes:

- Figura: Es todo lo contenido por el objeto **figure** de Matplotlib.
- Título: El título que se mostrará con la gráfica al dibujarla.
- Leyenda: Leyenda de la gráfica que mostrará los colores y nombres de las diferentes representaciones mostradas, líneas, puntos, etc.
- Cuadrícula: Línea que mostrará una cuadrícula por cada uno de los valores de los marcadores mayores.
- Lomo: El lomo de la gráfica son las líneas que conforman los límites de las gráficas representadas.
- Ejes X e Y: Los ejes de las gráficas, sobre los cuales se mostrarán los marcadores mayores y menores.
- Etiqueta Eje X e Y: Muestra el nombre de la variable representada en cada eje.
- Marcador mayor: Muestra la división entre los valores que alcanzan las marcas de los datos.
- Marcador menor: Muestra las subdivisiones entre cada una de las marcas mayores.
- Etiqueta de marcador mayor y menor: Muestra el valor numérico asociado a cada marcador, ya sea el mayor o el menor.
- Línea: Representación gráfica de los valores de una gráfica lineal.
- Marcador: Puntos correspondientes a una gráfica de tipo dispersión. Se han explicado las diferentes tipos de gráficas:
- Tipos de gráficas
 - Básicas
 - Plot: Representamos líneas, utilizando una serie de puntos. plt.plot()
 - Gráficas de barras:
 - Diagrama de barras: plt.bar(x,y)
 - Diagrama de barras horizontal: plt.barh(x,y)
 - Diagrama de barras apiladas: plt.bar(x,y1), plt.bar(x,y2)
 - Gráficas de dispersión: Representa los valores de dos pares de variables numéricas. plt.scatter()
 - Estadística
 - Histograma: Muestra la frecuencia en la distribución de los datos. plt.hist()
 - Diagrama de cajas: Nos muestra la mediana, cuartiles, outliers, etc. plt.boxplot()
 - Gráfico de sectores: plt.pie(x) El área de cada trozo del gráfico es proporcional a la frecuencia.
 - Violin plot: plt.violinplot(x) Similar al boxplot y muestra adicionalmente la frecuencia de los datos.

Pair Programming Visualización I

Es el momento de ponernos manos a la obra con la visualización de datos. En este primer ejercicio de *pair programming* trabajaremos con el csv que generamos en el ejercicio de *pair programming* de Pandas III, ese en el que habíamos eliminado algunas columnas de nuestro csv.

Hoy realizaremos una serie de gráficas usando la librería de `matplotlib` para familiarizarnos un poco con los datos.

Para poder solucionar los ejercicios de hoy no solo tendremos que visualizar, también tendremos que usar métodos que hemos aprendido en lecciones anteriores de Pandas como por ejemplo `value_counts()`, `isnull`, `describe`, etc.

Tendréis que realizar gráficas para contestar a las siguientes preguntas:

1. ¿Cuál es la frecuencia de cada una de las categorías de la columna `sex` ?
2. ¿Cuál es el porcentaje de nulos por columna? Usa un *pie chart* donde incluyáis el porcentaje de cada variable en el interior del quesito. Para eso tendréis que usar el parámetro `autopct`, [aquí](#) algo de documentación.
3. ¿Cuál es el número total de valores únicos de cada una de las columnas categóricas del *DataFrame*?
4. Filtrad los datos para quedarnos solo con los datos de USA.
 - ¿Cuántos ataques de tiburones hubo en USA a lo largo del tiempo? Muestra solo los que hayan ocurrido del 2000 en adelante.
 - En función del tipo, ¿cuántos ataques de tiburones de cada tipo hubo en USA? Muestra los resultados en un gráfico de línea.
5. Filtrad los datos para Spain:
 - ¿Cuántos ataques de tiburones hubo en España a lo largo del tiempo? Muestra solo los que hayan ocurrido del 2000 en adelante.
 - En función del tipo, ¿cuántos ataques de cada tipo de tiburones hubo en España? Muestra los resultados en un gráfico de línea.

Happy coding

Visualización II - Personalizando Matplotlib

¿Como podemos personalizar las gráficas en Matplotlib? Aprenderemos como modificar diversos atributos de las gráficas con el fin de que sean más explicativas y estéticamente más llamativas.

Personalización en Matplotlib

Las gráficas que hemos creado hasta ahora mostraban únicamente los valores que queríamos representar, sin mostrar las etiquetas de los ejes, las leyendas, o los títulos, entre otros. En esta lección vamos a ver cómo personalizar nuestras gráficas de Matplotlib, para poder explotar todo el potencial de las mismas. Ya que cuando se realiza una representación, siempre se busca que sea lo más informativa y fácil de comprender posible. Vamos a ello!



Visualizacion-2.ipynb 521KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen Visulización II - Personalización en Matplotlib

Parametros de personalización:

- Tamaños de las gráficas: Dentro del argumento plt.figure(figsize =(ancho, alto)). Las unidades de el tamaño de la imagen es en pulgadas.
- Ejes:
 - Titulo: plt.title(label='nombre_grafica'). No es necesario escribir el argumento label
 - Etiquetas de los ejes:
 - Etiqueta del eje X: ax.set_xlabel(xlabel = 'nombre_eje_x')
 - Etiqueta del eje Y: ax.set_ylabel(ylabel = 'nombre_eje_y')
 - Límites de los ejes: Limitar la zona de representación de la gráfica.
 - Eje X: plt.xlim([,])
 - Eje Y: plt.ylim([,])
 - Cuadrícula: Añade una cuadrícula al fondo de la gráfica, util cuando tenemos datos en diferentes escalas. ax.grid()
 - Color. color = "
 - Tipo de línea. linestyle = "
 - Tamaño. linewidth = "
 - Los ejes sobre los que queremos mostrar la cuadrícula
 - Etc
 - Leyenda. ax.legend(labels = ['label1','label2',....])
 - Marcadores: Iconos que representan los datos a mostrar en la gráfica
 - Tipos de marcadores

Marcador	Descripción
"."	Punto
Pixel	
"o"	Cirulo
"v"	Triángulo abajo
"^"	Triángulo arriba
"<"	Triángulo izquierda
Triángulo derecha	
"8"	Octágono
"s"	Cuadrado
"p"	Pentágono
"P"	Más (relleno)

"*"	Estrella
"h"	Hexágono 1
"H"	Hexágono 2
"+"	Más
"x"	x
"X"	x (relleno)
"D"	Diamante
"d"	Diamante fino

- Tamaño del marcador. `markersize = "`
- `color: c (o color) = 'nombre-color'`
- Multigráficas: Nos permite representar más de una gráfica dentro de un mismo objeto `Figure()`.
`plt.subplots((numero_filas, numero_columnas), figsize= (ancho,alto))`
- Guardar gráficas: Para guardar las gráficas en ficheros externos.
`plt.savefig(nombre_de_la_figura.extension)`
 - Las extensiones pueden ser:
 - jpeg
 - jpg
 - png
 - tiff
 - otras

Pair Programming Visualización II

Sigamos con más visualizaciones. En este caso trabajaremos con el csv que generamos en el ejercicio de *pair programming* de Pandas V, ese en el que habíamos unido distintos csv en uno.

El objetivo de hoy, mejorar algunas de las gráficas que hicimos en el ejercicio de *pair programming* de ayer y crear algunas nuevas aprovechando que tenemos algunas columnas nuevas en *dataframe*

Las preguntas que tendréis que contestar son:

1. Ayer creamos un par de gráficas para USA y España, donde visualizamos el número de ataques en cada país a lo largo de los años y el tipo de los ataques. Hoy tendréis que hacer dos gráficas con *subplots*:
 - En la primera debéis mostrar los tipos ataques en USA y España, cada subplot corresponderá a un país. Debéis:
 - Poner nombre a los ejes de cada uno de los subplots
 - Poner un títulos a cada uno de los subplots
 - Poner el color de las líneas en negro.
 - Establecer el tamaño de la gráfica a 15 x 5
 - Ponerle marcadores a las gráficas
 - No incluir leyenda
 - En la segunda debéis mostrar los ataques por año para cada uno de los países.
 - De nuevo tendréis que ponerle nombre a los ejes y título a cada gráfica.
 - En este caso el subplot deberá tener dos filas y una columna
 - No incluir leyenda
 - Cambiar el color de las barras
2. Cread un único gráfico donde juntéis las gráficas de los tipos de ataques en una sola. Debéis:
 - Poner leyenda a la gráfica
 - Cada línea debe ir en un color
 - Ponerle nombre a los ejes y a la gráfica.
 - Poner marcadores a las líneas.
3. Guardad las figuras creadas en cada caso. Podéis usar el formato que queráis. Guarda todas las fotos en una carpeta llamada *Visualizacion-II*.

Happy coding

Visualización III - Seaborn

¿Qué es Seaborn? Comprenderemos como utilizar este otro paquete de visualización de gráficos de Python, veremos como está relacionado con Matplotlib y aprenderemos como realizar algunos tipos de gráfi

Visualización III - Seaborn

Seaborn es, al igual que Matplotlib, uno de los paquetes que no puedes perderte cuando estás aprendiendo a realizar visualizaciones de conjuntos de datos. Ya que está completamente integrado con Pandas y se sienta sobre las bases de Matplotlib, para obtener de forma más sencilla gráficas haciendo uso de las características propias del tratamiento de datasets con Pandas.

En este [enlace](#) tenéis la página oficial de la librería Seaborn.



Visualizacion-3.ipynb 1MB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen Visualización III - Seaborn

- Seaborn es una librería para gráficos estadísticos que está completamente integrada con **Pandas**.
 - Consideraciones de uso:
 - Conjunto de datos que se va a utilizar, tiene que ser de tipo pandas dataframe.
 - Variables que queremos representar del dataframe.
 - Gráficas básica
 - Lineal: Representa una gráfica de tipo lineal.
 - `sns.lineplot(x = 'variable_x', y = 'variable_y', data = nombre_dataset, ci= None, hue='opcional' (variable categórica))`
 - Dispersion: Representa el diagrama de dispersión de las variables escogidas. - `sns.scatterplot(x = 'variable_x', y = 'variable_y', data = nombre_dataset, hue='opcional' (variable categórica))`
 - Histogramas: Representa el histograma de la variable seleccionada
 - `sns.histplot(x = 'variable_x', data = nombre_dataset, , hue='opcional' (variable categórica), kde = 'opcional', bins= 'num_bins')`
 - Diagrama de cajas: Representa el diagrama de caja de la variable seleccionada
 - `sns.boxplot(x = 'variable_x',y='variable_y'(puede ser categórica) data = nombre_dataset, hue='opcional' (variable categórica))`
 - Graficas avanzadas
 - Diagrama de enjambre: Representa un diagrama de dispersion, sin superponer los puntos para una variable categórica frente a una variable numérica.
 - `sns.swarmplot(x = 'variable_x', y = 'variable_y', data = nombre_dataset, hue='opcional')`
 - Countplot: Representar un histograma utilizando variables categóricas.
 - `sns.countplot(x = 'variable_x',y = 'variable_y', data = nombre_dataset, hue='opcional' (variable categórica))`
 - Catplot: Relaciona una variable categórica con una variable numérica.
 - `sns.catplot(x = 'variable_x', y = 'variable_y', data = nombre_dataset, hue='opcional')`
 - Pairplot: Representa el histograma y diagramas de dispersion de todas las variables numéricas con el resto de variables.
 - `sns.pairplot(data = 'nombre_dataset')`
 - Arg: Kind= {scatter, kde, hist, reg}

Pair Programming Visualización III

En este ejercicio, de nuevo, trabajaremos con el csv que generamos en el ejercicio de *pair programming* de Pandas V, ese en el que habíamos unido distintos csv en uno.

En estos ejercicios aprenderemos a hacer algunas gráficas usando la librería `seaborn`. Igual que el ejercicio anterior de *pair programming*, para solucionar algunas preguntas tendréis que usar métodos y herramientas aprendidas anteriormente como `groupby`, `value_counts`, `rename`, filtrado de datos, etc.

1. Usando la librería `seaborn`, crea una gráfica de barras que muestre el número de personas que han hecho cada actividad en cada año. Hacedlo solo desde el 2015. Os podrá salir una gráfica un poco fea, no os preocupeis, es lo que esperamos.
2. Usando la librería `seaborn`, crea una gráfica de barras que muestre el número de ataques que han ocurrido cada año en función del sexo. Hacedlo solo desde el 2015.
3. ¿Cuántos ataques hubo por año? Seleccionad solo desde el año 1500.
4. Seleccionad solo los registros de USA, AUSTRALIA y SOUTH AFRICA. Muestra el número de ataques por país y año desde el año 1900.
5. Realizad un `boxplot` donde se muestre los principales estadísticos del número de ataques por país.

NOTA Somos analistas y no nos vale con sacar las gráficas. Includ una celda de *markdown* donde expliqueis que es lo que estamos viendo en las gráficas, es decir, haced una interpretación de los resultados.

Happy coding

Estadística

Estadística I - Introducción

¿Qué es la estadística? En esta lección presentamos los conceptos más básicos de la estadística descriptiva y cómo calcular algunos de los estadísticos más comunes usando Python.

Conceptos estadística I

Para una *Data Analyst* tener nociones de probabilidad y estadística es fundamental. En los últimos años, la explosión del aumento de datos disponibles ha hecho que la necesidad de extraer información útil de los mismos sea más y más importante. Python es una herramienta muy útil a la hora de explorar los datos y obtener conocimiento.

¿Qué es la estadística?

Según la Wikipedia y otros sitios de Internet: "La estadística es la rama de la matemática que estudia la variabilidad, colección, organización, análisis, interpretación, y presentación de los datos, así como el proceso aleatorio que los genera siguiendo las leyes de la probabilidad". Puede que esta frase no signifique demasiado para nosotras de momento, así que simplemente quedémonos con que saber acerca de estadística es una de las habilidades más necesarias para una *Data Analyst* ya que su trabajo está íntimamente relacionado con los datos: su recolección, análisis e interpretación.

Los datos sin procesar esconden una gran cantidad de información en su interior y es la tarea de una *Data Analyst* el acceder a esa información y entenderla, en gran parte mediante el uso de la estadística. Podemos dividir la estadística en dos grandes ramas dependiendo de su objetivo:

- *Estadística descriptiva*: es la rama que se dedica a recolectar, ordenar, analizar y representar los conjuntos de datos, con el fin de describirlos de una manera lo más precisa posible. Es decir, aportar información acerca de los datos reales.
- *Estadística inferencial o predictiva*: es la rama que intenta sacar conclusiones generales para toda la población a partir del estudio de una muestra. Es decir, intenta predecir futuros comportamientos de los datos en base a un conjunto previo.

En esta primera lección nos centramos en la rama descriptiva de la estadística, calculando parámetros como la media, mediana, desviación típica, percentiles, etc. En lecciones más avanzadas, tocaremos la parte inferencial de la estadística centrándonos en la regresión, clasificación y clustering de datos nuevos, utilizando el conocimiento adquirido de los datos anteriores.



estadistica-I.ipynb 34KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Estadística I - Introducción

- Estadística descriptiva: Uso de medidas para describir las propiedades de conjuntos de datos.
 - Librerías más utilizadas para cálculo de estadísticos en Python:
 - Numpy
 - Scipy métodos de stats
 - Pandas
 - Matplotlib
 - Seaborn
 - Estadísticos básicos:
 - Media (aritmética): Suma de todos los elementos de un conjunto de datos dividido entre el número total de elementos del mismo conjunto de datos.
 - Desviación respecto a la media: Diferencia en valor absoluto entre el valor de cada dato y su media.
 - Varianza: La media aritmética al cuadrado de la desviación respecto de la media.
 - Desviación típica: La raíz cuadrada de la varianza.
 - Media ponderada: Es un tipo de media aritmética donde a cada valor del conjunto de datos se le asigna un peso que afectará a la contribución de cada uno de los valores del conjunto de datos.
 - Moda: Es el valor con mayor frecuencia del conjunto de datos.
 - Mediana: Es el valor central donde tendremos el 50% de los datos por encima de este valor y el 50 % por debajo. Para ello es necesario tener el conjunto de datos ordenados de menor a mayor y no tiene que ser un dato que exista en el conjunto de los datos.
 - Mínimo y máximo: Valor mínimo y máximo del conjunto de datos.
 - Robustez: Medida que se utiliza para ver la resistencia de nuestros estadísticos cuando los datos son muy diferentes entre sí.
 - Coeficiente de variación: Se obtiene haciendo el cociente entre la desviación típica y la media. Nos dice si los datos son homogéneos o heterogéneos.

Pair Programming Estadística I

Empezamos una nueva lección, y este es el momento de poner en práctica los conocimientos adquiridos del *Estadística Descriptiva básica*. En este caso vamos a darle un giro de tuerca a la forma en la que se ha resuelto los ejercicios de la *Pair* de hasta ahora! Para ello os vamos a pedir que por vuestra cuenta busqueis un dataset que os guste, por ejemplo de Kaggle o de donde queráis extraerlo. De esta forma vamos a estudiar los diferentes estadísticos que contiene en dataset seleccionado **Para todas los ejercicios de pair programming de Estadística I, II y III usaremos el dataframe que se tome para el desarrollo de este ejercicio de pair programming.**

Objetivos

Los objetivos de hoy son:

1. Buscar un conjunto de datos a analizar:

- Se recomienda que el conjunto de datos a analizar tenga variables numéricas y categóricas. Ya que también se analizarán.

2. Extraer los siguientes estadísticos del conjunto de datos para varias de las variables disponibles:

- Media
- Desviación con respecto de la media
- Varianza
- Desviación típica
- Moda
- Mediana
- Media ponderada
- Robustez
- Coeficiente de variación

3. Interpretación de los resultados.

- Ahora interpreta los resultados obtenidos de los diferentes estadísticos estudiados haciendo hincapié en aspectos interesantes de tus datos y explica lo mejor posible la naturaleza de los mismos.

Happy coding

Estadística II - Cuartiles, estadística con pandas y tabla de frecuencias

¿Qué es la estadística? En esta lección presentamos los conceptos más básicos de la estadística descriptiva y cómo calcular algunos de los estadísticos más comunes usando Python.

Conceptos estadística II

Vamos a introducir ahora unos últimos conceptos básicos de estadística descriptiva para luego comenzar con algunos aspectos de estadísticos ligeramente más complejos como las tablas de frecuencias.



estadistica-II.ipynb 67KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Estadística 2

Percentiles y Quartiles

- **Percentil:** un valor para identificar como estan distribuidos los datos. Están ordenados de menor a mayor, es una medida relativa ya que depende de los valores que tome nuestra variable, la podemos a entender como una medida de posición. Dividimos en 100 partes nuestros datos.
- **Quartil** es lo mismo que el percentil pero dividimos el conjunto de datos en 4 partes. Podemos identificar:
 - Q1: es el 25%. Asumiendo que el valor del Q1 es de 56, decimos que el 25 % de mis datos tienen un valor menor que 56. Es lo mismo que el percentil 25%
 - Q2: es el 50% que es lo mismo que la mediana. Asumiendo que el valor del Q2 es de 67, decimos que el 50% de los datos tiene un valor menor que 67 o mayor que 67. Que es lo mismo que el percentil 50
 - Q3: es el 75%. Asumiendo que el valor del Q3 es de 100, decimos que el 75% de los datos tienen un valor menor que 100.
- **Rango intercuartílico:** nos sirve para poder calcular el tamaño de los bigotes.
- **Bigotes** nos ayudan a indentificar los valores atípicos, ya que todos los puntos que estén más allá de estos bigotes podrán ser considerados como valores atípicos.
- **Datos atípicos:** datos extremos que se encuentran muy alejados de la mediana o del Q1 o Q3. Estos son de especial interés, ya que pueden implicar que son datos incorrectos o datos interesantes de estudiar, como se alejan del resto de los datos.
- Para calcular los percentiles usaremos el método de NumPy `np.percentile`. Pero... **si tenemos nulos en nuestra variable usaremos:**

```
np.nanpercentile()
```

Tablas de frecuencias

- Tablas de frecuencias absolutas
- Tablas de frecuencias relativas
- Tablas de frecuencias absoluta acumulada
- Tablas de frecuencias relativas acumulada
- La diferencia entre la absoluta y la relativa es que la absoluta es el número de veces que aparece en el total de mis datos y la relativa

Pair Programming Estadística II

Vamos ahora a tomar de nuevo el conjunto de datos con el que estuvimos trabajando ayer y vamos a continuar con el análisis de las estadísticas de los contenidos del conjunto de datos

Para todas los ejercicios de *pair programming* de Estadística I, II y III usaremos el *dataframe* que se tome para el desarrollo de este ejercicio de *pair programming*.

Se ruega a la hora de realizar la entrega que incluyáis el conjunto de datos que hayáis decidido emplear para estos ejercicios.

Objetivos

Los objetivos de hoy son:

1. Extraer los siguientes estadísticos del conjunto de datos para varias de las variables disponibles:
 - Utilizando métodos de pandas: media, mediana, moda, máximo y mínimo
 - Percentiles 25, 75
 - Rango Intercuartílico
 - Boxplot de algunas variables
 - Tabla de frecuencias
 - Variables cualitativas
 - Variables cuantitativas
 - Frecuencia acumulada
 - Representación de al menos una tabla de frecuencias
2. Interpretación de los resultados.
 - Ahora interpreta los resultados obtenidos de los diferentes estadísticos estudiados haciendo hincapié en aspectos interesantes de tus datos y explica lo mejor posible la naturaleza de los mismos.

Happy coding

Estadística III - Tablas de contingencia, correlación, sesgos y intervalos de confianza

Después de introducir los conceptos más básicos de estadística descriptiva en las secciones anteriores, en esta vamos a continuar profundizando en cálculos algo más avanzados como las tablas de correlación y autocorrelacion de variables, sesgos en los datos, e intervalos de confianza.



estadistica-III.ipynb 114KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Estadística III

Vamos ahora a tomar de nuevo el conjunto de datos con el que estuvimos trabajando en los últimos días y vamos a continuar con el análisis de las estadísticas de los contenidos del conjunto de datos.

Para todas los ejercicios de *pair programming* de Estadística I, II y III usaremos el *dataframe* que se tome para el desarrollo de este ejercicio de *pair programming*.

Se ruega a la hora de realizar la entrega que incluyais el conjunto de datos que hayais decidido emplear para estos ejercicios.

Objetivos

Los objetivos de hoy son:

1. Extraer los siguientes estadísticos del conjunto de datos para varias de las variables disponibles:
 - Tabla de contingencia
 - Coeficiente de correlación de las variables numéricas del dataset.
 - Sesgos de alguna de las variables numéricas del dataset.
 - Intervalo de confianza de algunas de las variables numéricas del dataset
2. Interpretación de los resultados.
 - Ahora interpreta los resultados obtenidos de los diferentes estadísticos estudiados haciendo hincapié en aspectos interesantes de tus datos y explica lo mejor posible la naturaleza de los mismos.

Happy coding

Repaso Conceptos Estadística y Visualizacion

Repaso Conceptos Resumidos

Lección de repaso de conceptos básicos de estadística y visualización

En esta lección volveremos sobre conceptos que hemos aprendido previamente, en las lecciones de Visualización y Estadística.



housing_melbourne.csv 7MB

Binary

Descarga este fichero y cargalo en VS Code.



repaso_esta_viz.ipynb 667KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Repaso Conceptos Resumidos

Al igual que en los ejercicios de *pair programming* de estadística, deberéis elegir un set de datos, puede ser de cualquier tipo. El objetivo de este *pair programming* es:

- Hacer una exploración inicial del *dataframe*.
 - ¿Cuántas filas tiene?
 - ¿Cuántas nulos tiene?
 - ¿Y duplicados?
 - ¿Qué tipo de datos tenemos en el *dataframe*?
 - etc.
- Plantead una serie de preguntas como las que fuimos contestando en la clase invertida.
- Contestad a las preguntas planteadas usando los principales estadísticos que hemos aprendido y gráficas que mejor se ajusten.

Happy coding

EDA

Datos-EDA

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de EDA

En esta apartado encontrareis los ficheros que necesitaréis para las lecciones de EDA:

- **Sesión teórica**

- `students.csv`



`students.csv` 62KB

Binary

Descarga fichero para poder seguir la clase invertida.

- **Ejercicios prácticos**

- `netflix_titles.csv`



`netflix_titles.csv` 2MB

Binary

Descarga este fichero para poder hacer los ejericicios que os planteamos.

EDA I - Introducción

¿Qué es el EDA? ¿Cómo realizar un análisis exploratorio de datos?

EDA I - Introducción

El *análisis exploratorio de datos* (EDA) es utilizado por los analistas de datos para analizar e investigar conjuntos de datos y resumir sus principales características, empleando a menudo métodos de visualización de datos.

Nos va a ayudar a identificar la mejor manera de manipular los datos para obtener los mejores resultados para las preguntas que nos hacemos, lo que nos va a permitir descubrir patrones, detectar anomalías, probar una hipótesis o comprobar supuestos.

En esta lección, haremos una primera aproximación al análisis exploratorio de nuestros datos usando métodos de Pandas que ya conocemos. Estos métodos incluyen entre otros:

- `head()` o `tail()`
- `describe()` e `info()`
- `isnull()`
- `unique()` y `value_counts()`



EDA-I-Intro_EDA.ipynb 41KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen EDA I - Introducción

EDA es el Análisis Exploratorio de Datos Sirve para hacer investigaciones iniciales, es decir para cotillear en mis datos y saber con qué estamos trabajando.

- Métodos útiles:
 - `df.describe()` : describe los valores numéricos, con media etc
 - `df.describe(include='object')` : describe la categoría más frecuente y su frecuencia, por columna categórica
 - `df.info()` : devuelve por columna el tipo de datos y la cantidad de entradas, es decir, la cantidad de no-nulos
 - `df.head()` y `df.tail()` : devuelve las primeras o últimas entradas del dataframe
 - `df.isnull()` :devuelve un True para cada nulo, y el resultado es una serie evaluando los nulos de toda la columna.
 - `df.isnull().sum()` :cuenta todos lo True de mi serie
 - `df.unique()` :son los valores únicos por columna
 - `df.value_counts()` :devuelve los valores únicos con su frecuencia por columna
- Propiedades del dataframe: (no llevan paréntesis)
 - `df.shape` :devuelve la forma del dataframe (num_filas, num_columnas)
 - `df.dtypes` :devuelve el tipo de datos de esa columna
 - `df.T` :devuelve el transpuesto, cambia las columnas a filas y vice versa.

Pair Programming EDA I

Empezamos una nueva lección, y este es el momento de poner en práctica los conocimientos adquiridos del *Análisis Exploratorio de Datos*. Si recordamos, en la lecciones de Pandas estuvimos usando un *dataset* sobre ataques de tiburones, y que hicimos algunos cambios a lo largo de los ejercicios, incluimos filas, columnas e incluso limpiamos algunas columnas ! **Para todos los ejercicios de pair programming de EDA I y II usaremos el *dataframe* generado en el ejercicio de Pandas VII.**

En el ejercicio de hoy nos familiarizaremos con el *dataset* completo sobre el que trabajaremos en la parte de limpieza.

Pero para hacer una exploración buena del *dataset* no lo podemos hacer como pollo sin cabeza a ver que nos encontramos. Necesitamos hacernos una serie de preguntas o hipótesis que queramos contestar. En este caso os plantemos las siguientes preguntas a contestar a lo largo de los ejercicios de *pair programming* de este módulo:

Hipótesis

Las hipótesis que os planteamos son:

- ¿Es Australia el sitio más peligroso y letal para estar relajada en la playa?
- ¿Hay diferencias entre los países en los ataques a hombres y mujeres?
- ¿Cuáles son las edades que más sufren ataques?
- Independientemente de la edad, ¿sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

Objetivos

Los objetivos de hoy son:

- Haced una exploración inicial del *dataframe*:
 - ¿Cuál es el número de filas y columnas?
 - ¿Cuál es el porcentaje de valores nulos en cada columna? Presenta los resultados en un *dataframe*, donde las columnas sean el nombre de las variables y el porcentaje de nulos.
 - ¿Qué tipos de datos hay en cada columna?
 - Extraed la información general del *dataframe*.
- Cread dos *dataframes* nuevos, uno que incluya solo las variables categóricas y otro que incluya solo las variables numéricas.
 - ¿Cuáles son los principales estadísticos de los *dataframes*?
 - Extraed los valores únicos su frecuencia. **BONUS** presenta los resultados en un *dataframe*. Os deberá quedar algo como esto:

variable	n_uniques
case_number	[2018.06.25, 2018.06.18, 2018.06.09, 20...
type	[Boating, Unprovoked, Invalid, Provoked,
country	[USA, AUSTRALIA, MEXICO, BRAZIL, ENGLAND, SOUT...

Donde `variable` es cada una de las variables categóricas que tenemos en el *dataset* y `n_unicos` es una lista con los nombres de los valores únicos de cada variable.

Conclusiones

De qué nos sirven estos números y *dataframe*? Extraed conclusiones de ellos.

Por ejemplo:

- ¿Hay alguna columna que no tenga sentido su tipo de datos?
- ¿Hay alguna columna que no tenga sentido su nombre?
- ¿Hay alguna columna que no tenga sentido su contenido?
- ¿Hay alguna columna que tenga demasiadas categorías o valores que nos pueda hacer difícil trabajar con ella?

Happy coding

EDA II - Nulos y Valores extremos

Aprenderemos como hacer un análisis descriptivo de nuestros datos en profundidad. Además, veremos qué son los valores nulos y los outliers. ¿Qué son? ¿Por qué son importantes? ¿Cómo los detectamos?

EDA II - Nulos & Outliers

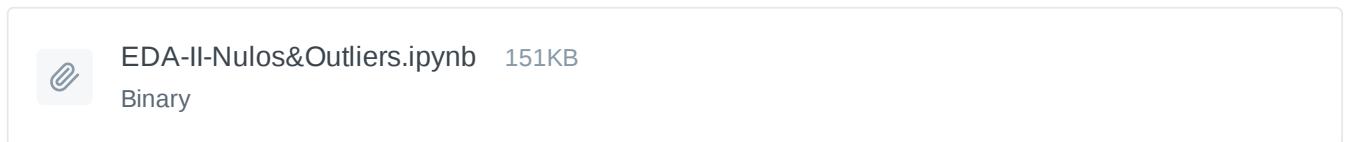
Como hemos visto, el análisis exploratorio tiene como objetivo identificar y entender nuestros datos. Dicho análisis se basa en gráficos y estadísticos que permiten explorar la distribución identificando características tales como: valores atípicos o outliers, saltos o discontinuidades, concentraciones de valores, la forma de la distribución, etc.

En esta segunda sesión nos centramos en la parte de descripción estadística básica de nuestros datos.

Por otro lado, veremos qué son los **valores nulos** o **valores perdidos**. Normalmente los veremos como `NaN`, pero habrá ocasiones en las que los valores nulos se representan con un valor diferente, por ejemplo, con un `00000` en el caso de una variable numérica o `"NaN"` o `"n/a"`, en el caso de variables de tipo *string*. En este último caso, lo que debemos hacer es reemplazar el valor de estos valores perdidos por el valor `NaN`. Veremos qué nulos nos podemos encontrar y cómo identificarlos.

Por último, veremos qué son los *outliers*. ¿Por qué nos interesa detectar esos outliers? Porque pueden afectar considerablemente a los resultados de nuestras conclusiones. Para mal... o para bien! Por eso hay que detectarlos, y tenerlos en cuenta.

[Aquí](#) tenéis una pequeña infografía de qué es un proceso de EDA, explicado para todos los públicos.



Repaso

- Los NaN (not a number) son de tipo Float64. Es el tipo de nulos de numpy, np.nan. Otro tipo de nulo es el None. Todas las instancias de None son el mismo None, apuntan al mismo sitio en la memoria, importante para comparaciones como el if. Los NA (not a number) son de tipo float.
- *Outliers* son valores anormales, que tienen una definición.

`Q1 - 1.5 * IQR`

`Q3 + 1.5 * IQR`

- La desviación estándar, std, en una distribución normal me dice cuánto en general se desvían los valores de la media.
- La librería sidetable hace tablas de frecuencias y valores nulos.
 - Al utilizar stb.freq con value (para que nos ordene los datos en función de otra columna), los porcentajes de las frecuencias pueden variar, esto depende de los nulos que tengamos en la columna que incluimos en "value".
- El `missing` de sidetable y el `isnull().sum()`, son similares. La diferencia es que con `missing` de sidetable nos sale por defecto el porcentaje de nulos.

Pair Programming EDA II

En esta lección hemos aprendido qué son los valores nulos y los *outliers* o valores faltantes. Además hemos aprendido una librería super molona que nos permite hacer una exploración de nuestro *dataset* muy profunda.

Hipótesis

Antes de seguir, recordamos las preguntas que nos planteamos al principio del *pair programming* de EDA para dirigir nuestro análisis.

- ¿Es Australia el sitio más peligroso y letal para estar relajada en la playa?
- ¿Hay diferencias entre los países en los ataques a hombres y mujeres?
- ¿Cuáles son las edades que más sufren ataques?
- Independientemente de la edad, ¿sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Reducimos esto a una tabla para saber qué tenemos ya limpio y qué no. Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✗ esta en formato <i>string</i> cuando debería ser <i>integer</i> , y en algunos casos tenemos rangos de edad
species	✗ es un jaleo! Debemos unificar los nombres y reducir a las especies más importantes
country	✗ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas V
year	✗ es una columna de tipo <i>float</i> , deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas V
fecha	✓ la limpiamos en el <i>pair</i> de Pandas V

Objetivos

Usad la librería `sidetable` para:

1. Explorar los valores nulos.
2. Explorad la columna de `country` y reportad los resultados solo para aquellas categorías que incluyan el 70% de los datos.
3. Explorad la columna `country` en relación al sexo, filtrando de nuevo por aquellas categorías que incluyan el 70% de los datos.
4. Haced lo mismo para la columna `activity` en relación a la columna `fatal`. En este caso filtrad y mostrad los datos que incluyan el 60% de los datos.
5. Por último, realizad este análisis para la columna `fecha` (la que creamos en el último ejercicio de *pair programming* de Pandas) en relación a la columna `fatal`. Mostrad solo los resultados para las categorías que incluyan el 60% de los datos.

Nota Igual que siempre, no nos vale solo con picar el código, queremos que nos incluyais las conclusiones que extraéis.

Conclusiones

Si bien es cierto que la lección de hoy ha sido muy teórica, en esta parte los ejercicios no serán tanto de programar sino de pensar . ¿Qué tendréis que hacer?

- En relación a los valores nulos:
 - Identificar las columnas que tienen nulos (lo hicimos en el ejercicio de *pair programming* EDA I y en el de hoy con el `sidetable`).
 - Debatid entre vosotras qué haríais con los valores nulos, ¿los quitaríais o los reemplazaríais con un valor que nosotras decidiremos?. Cread una celda de *markdown* en vuestro jupyter y escribid vuestras conclusiones.
- En relación a los *outliers*, ya hemos ido conociendo un poco nuestro *dataframe* y de momento solo tenemos una columna de tipo numérico, por lo tanto, poco podemos hacer. Un poco más adelante, cuando hayamos hecho una serie de cambios sobre nuestras columnas podremos hacer una evaluación más profunda de estos.

Happy coding

EDA III - Análisis exploratorio gráfico

Aprenderemos como familiarizarnos con los datos que tenemos de una forma visual. Como son las relaciones entre las variables, ya sean numéricas o categóricas y lo más importante, como interpretarlas.

EDA III - Visualización

En análisis de datos existen muchos tipos diferentes de datos para analizar. La visualización de los datos tiene un papel fundamental en todos las fases del análisis de datos.

La visualización de datos está en el núcleo del proceso. Es una herramienta básica para el analista o científico de datos que -mediante un proceso iterativo- va transformando y componiendo un modelo lógico de los datos. Apoyándose en la visualización, el analista va descubriendo los secretos enterrados en los datos.

La visualización permite de forma rápida:

- Descartar aquellos datos poco representativos o erróneos.
- Identificar aquellas variables que dependen unas de otras y por lo tanto contienen información redundante.
- Realizar cortes a los datos para poder observarlos desde diferentes perspectivas.
- Comprobar que aquellos modelos, tendencias, predicciones y agrupaciones que hemos aplicado sobre los datos, nos devuelven el resultado esperado.



EDA-III_Analisis-Exploratorio-Grafico.ipynb 2MB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso

- Relación numéricas con numéricas
 - `Scatterplot` : es una gráfica de dispersión en que solapan los datos y que ayuda a detectar la correlación, patrones de tendencia o valores atípicos.
 - `Regplot` : un `scatterplot` con una línea de regresión encima.
 - `Heatmap`
 - Creamos "mask" para darle estilo a nuestro heatmap.
 - `np.triu` nos crea un array triangular para poder cambiar el mask de nuestro heatmap.
 - Lo creamos con una matriz del mismo tamaño que `corr()`, a la que especificamos el tipo de dato (booleano).

```
mask = np.triu(np.ones_like(df.corr(), dtype=np.bool))

sns.heatmap(df.corr(),
            cmap = "YlGnBu", # para cambiar el color
            mask = mask,
            annot = True);
```

- `Jointplot`: datos bivariados. Nos aparecerá en la misma gráfica un histograma y un `scatterplot`. Para interpretar ese gráfico, lo haremos por separado.
- Para saber como son nuestras variables categóricas
 - `Countplot`: nos mostrará cuántas filas hay de cada una de nuestras categorías
- Categórica y numérica
 - `Swarmplot`: tenemos la dispersión en valores numéricos por categoría.
 - `Boxplot`: nos muestra los principales estadísticos de nuestra variable
 - `Violinplot`: es como un `boxplot`. Nos mostrará la mediana con el puntito blanco y el rango intercuartílico con la líena negra gorda.
 - `Pointplot`: el intervalo de confianza de cada punto calculado por categoría, conectados entre sí.
 - Si queremos cambiar el intervalo de confianza mostrado en la gráfica usamos "ci".

```
sns.pointplot(x="gender", y="math score", data=df, ci = 99);
```

Pair Programming EDA III

Es el momento de hacer una buena exploración visual de los datos para entender un poquito mejor el *dataset*. Para ello trabajaremos con la librería `matplotlib` y `seaborn`.

Hipótesis

Antes de seguir, recordamos las preguntas que nos planteamos al principio del *pair programming* de EDA para dirigir nuestro análisis.

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Hay diferencias entre los países en los ataques a hombres y mujeres?
- ¿Cuáles son las edades que más sufren ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Reducimos esto a una tabla para saber qué tenemos ya limpio y qué no. Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I y II ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✓ la limpiamos en el <i>pair</i> de Limpieza II
species	✓ la limpiamos en el <i>pair</i> de Limpieza II
country	✓ la limpiamos en el <i>pair</i> de Limpieza II
fatal	✓ la limpiamos en el <i>pair</i> de Pandas V
year	✓ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas V
fecha	✓ la limpiamos en el <i>pair</i> de Pandas V

Objetivos

En el ejercicio de hoy tendremos que contestar a las siguientes preguntas:

Nota Como siempre, después de cada gráfica incluid una pequeña explicación de lo que nos está mostrando la gráfica.

1. Estableced el tamaño de las figuras al inicio de vuestro jupyter para que todas tengan las mismas dimensiones.
2. Manos a la obra con los plots, estos irán muy dirigidos hacia las preguntas que queremos contestar, para hacer una primera aproximación a nuestras hipótesis:
 - ¿Cuántos ataques hubo por pais? ¿Se puede ver algo claro si ploteamos todos los paises? En caso de que no, seleccionad solo los 10 paises con más ataques y volved a hacer el plot.
 - Usando el DataFrame de los 10 paises con más ataques, ¿hay diferencias entre sexos en esos paises?
 - ¿Qué edad recibe mayor número de ataques? **Pista** Para que esta gráfica nos salga legible tendremos que crear antes grupos de edad. Para hacer esto de una forma sencilla tenemos el método `pd.cut()` que aprendimos en la lección de Limpieza II.
 - Ataques entre sexos, ¿hay diferencia?
 - ¿Cuando ocurren más ataques? ¿Os sale algo raro en la gráfica? Intentad limpiar un poco los datos para que os salga mejor.

Podríamos hacer más gráficas para seguir explorando nuestras hipótesis, pero tenemos el tiempo limitado . Aún así, sentiros libres de explorar todo lo que queráis vuestros datos para entenderlos mejor .

Happy coding

Limpieza

Datos-Limpieza

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de Limpieza

En esta apartado encontrareis los ficheros que necesitaréis para las lecciones de Limpieza:

- **Sesión teórica**

- [WDICountry2.csv](#)



WDICountry2.csv 120KB

Binary

Descarga fichero para poder seguir la clase invertida.

- [WDICountry_limpio.csv](#)



WDICountry_limpio.csv 95KB

Binary

Descarga fichero para poder seguir la clase invertida.

- [steam.csv](#)



steam.csv 4MB

Binary

Descarga fichero para poder seguir la clase invertida.

- [steam_nulos.csv](#)



steam_con_nulos.csv 6MB

Binary

Descarga fichero para poder seguir la clase invertida.

- [house.csv](#)



house.csv 173KB

Binary

Descarga fichero para poder seguir la clase invertida.

- **Ejercicios prácticos**

- [south.csv](#)



south.csv 1MB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.

Limpieza I - Introducción

En este jupyter nos familiarizaremos con los primeros pasos en el proceso de limpieza de un DataFrame.

Del mismo modo que un cocinero antes de ponerse a cocinar necesita tener todos los ingredientes preparados (limpiar las verduras, pelar las patatas, adobar la carne, etc.), un analista de datos necesita realizar una serie de operaciones previas antes de ponerse a resolver el problema que está tratando, para hacer los datos “comestibles”.

Debido a la gran cantidad de datos que se generan, este primer paso suele ser de vital importancia para corregir múltiples deficiencias que podemos encontrar, y para ser capaz de extraer realmente la información relevante para nuestro problema. Según distintas encuestas realizadas a analistas de datos, en torno a un 80% del tiempo de trabajo se centra en obtener, limpiar y organizar los datos mientras que tan solo el 3% del tiempo se dedica a construir modelos de aprendizaje automático.

Hoy empezaremos a hacer unos primeros cambios en nuestros datos, preparándolos para poder responder a las preguntas que nos planteamos. Entre las herramientas que aprenderemos hoy podemos incluir:

- Eliminación de duplicados
- Cambio de nombre de columnas para unificarlas
- Eliminar columnas redundantes
- Cambiar el tipo de datos de nuestras columnas que a veces vienen mal. Por ejemplo números que vienen como *strings* o números enteros que vienen como decimales.



Limpieza-I-Basicos.ipynb 156KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

- Cambiar el nombre de las columnas: Para homogeneizar los nombres de las columnas y que sean iguales. Tenemos dos formas de hacerlo:
 - `.rename()` : en este caso tenemos que pasarle un diccionario que podremos crear con un dict comprehension


```
# creamos el dict comprehension
nuevas_columnas = {columnas: nueva_columnas.replace(' ', '')} for nueva_columnas in df.columns:
```

```
# aplicamos el método rename
df.rename( columns = nuevas_columnas , inplace = True)
```

```
# El atributo inplace, nos aplica los cambios sobre el dataset original.
```
 - `.columns` : le tenemos que pasar una lista


```
# creamos la lista
nuevas_columnas = [col.replace(" ", "_").lower() for col in df.columns]
```

```
# ejecutamos el cambio de nombre
df.columns = nuevas_columnas
```
- Colocar una columna como índice o cambiar el índice que tenemos:
 - Convertir una columna en indice(quitar columna de índice original)


```
df.set_index("index_" , inplace = True)
```
 - Cambiar el orden de las columnas


```
df.reindex(columns=nuevo_orden)
```
- El EDA y la limpieza de los datos van de la mano. Exploramos, para hacer una buena limpieza de los datos.
- Los duplicados, no nos gustan en ningún sitio. Debería ser de los primero que hacemos cuando empezamos a explorar/limpiar los datos.


```
df.duplicated() # esto nos chequea todo el df
```

```
df.duplicated([nombre_col]) # si quiero ver los duplicados para una columna dada
```
- Eliminación de duplicados:


```
df.drop_duplicates(inplace = True) # sobreescibirmos el df original
```

```
df_sin_duplicados = df.drop_duplicates() # genero un df nuevo sin duplicados
```
- Cambio de tipo de datos, podemos usar dos métodos:
 - `.astype()` : le tenemos que pasar el nuevo tipo de dato que queremos tener. Cosas importantes:
 - `copy` : por defecto es True, en este caso nos crea una copia de la columna a la que le queremos cambiar el tipo.
 - `errors` : para poder gestionar los errores. Dos opciones principales:
 - `ignore` : nos ignora la acción de cambio de dato. No nos dará error.

- `raise`: nos devuelve un error, avisándonos de que no puede hacer el cambio de tipo de datos que le estamos pidiendo.
- `pd.to_numeric(df[col])`: solo nos vale para convertir a numérico, tanto float como int. No tenemos que especificar el tipo de número que queremos
- Para filtrar las columnas en función del tipo de dato. Su sintaxis:

```
# para seleccionar las columnas de tipo object
df.select_dtypes(include = "object")
df.select_dtypes(include = "O")
```

```
# para seleccionar las columnas de tipo numérico
df.select_dtypes(exclude = "object")
df.select_dtypes(include = np.number)
```

- Cuando tenemos nulos, son de tipo `float`. Los nulos de pandas son `NaN` o `<NaN>`
- Si queremos cambiar el orden de las columnas en base a un orden alfabético podemos hacer lo siguiente:

```
# utilizando el método reindex
df.reindex(columns = df.columns.sort_values())

# utilizando el método .columns.
df.columns = df.columns.sort_values()

# el .sort_values() nos permite ordenar de menor a mayor o alfabéticamente los valores que
```

Pair Programming Limpieza I

Es el momento de empezar a limpiar nuestro *dataframe* que hemos ido viendo que está bastante sucio!

En este ejercicio trabajaremos con el *dataframe* que nos guardamos en la sesión de *pair* de Pandas VII.

Antes de seguir, recordamos las preguntas que nos planteamos al principio del pair programming de EDA para dirigir nuestro análisis.

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Cuál es el rango de edad que sufre la mayoría de los ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Si reducimos esto a una tabla para saber que tenemos ya limpito y que no:

variable	¿Está limpia?
age	✗ esta en formato <i>string</i> cuando debería ser <i>int</i> y en algunos casos tenemos rangos de edad
species	✗ es un jaleo! Debemos unificar los nombres reducir a las especies más importantes
country	✗ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas ✓
year	✗ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas ✓
fecha	✓ la limpiamos en el <i>pair</i> de Pandas ✓

Entre las tareas que tendréis que hacer hoy:

1. Al empezar a trabajar con este *dataframe* ya eliminamos algunas columnas que *a priori* no nos interesaban. Ahora llega el momento de eliminar alguna columna más. En este caso tendréis que eliminar las columnas que no nos sean útiles para contestar a nuestras preguntas. Pero ojo Δ , haced una copia del *dataframe* para no "cargarnos" el *dataframe* original y perder la info.
2. ¿Hay valores duplicados en nuestro *dataframe*? En caso de que los haya, eliminándlos.
3. Como hemos visto, algunas columnas no tienen el tipo de datos que deberían. Cambiad el tipo de dato para la columna de `year`.
4. En la columna de `country` poner todos los valores en minúsculas.
Pista Tendréis que usar una función o una `Lambda`
5. Guardamos el csv para seguir trabajando en el siguiente ejercicio de *pair* de limpieza.

Limpieza II - Strings

Aprendemos como trabajar y limpiar las columnas de tipo string.

Limpieza e Imputación II - Strings

Ya hemos empezado a cocinar y nuestro dataframe cada vez tiene mejor pinta. Uno de los principales problemas que nos podemos encontrar en los datos es que las columnas de tipo *string* están un poco sucias. Por ejemplo columnas con demasiada información que podemos separar en distintas columnas. O columnas que nos gustaría categorizar en distintos grupos. Por ejemplo, tenemos una columna con horas y queremos crear una nueva donde en vez de tener muchos valores solo tengamos tres: mañana, tarde, noche. Para hacer esto tenemos muchas aproximaciones, las que aprenderemos:

- Creación de intervalos, el ejemplo de la hora.
- Creación de porcentajes, podemos querer saber el porcentaje de ganancia de la venta de un producto.
- Creación de ratios, ratio entre dos variables. Δ La diferencia con los porcentajes es que en el caso de los ratios no multiplicaremos por 100. Será la división de las dos columnas.
- Además, en Pandas tenemos métodos propios para realizar este tipo de cambios
 - `applymap` : se aplicará a todo el *DataFrame*
 - `apply` : se aplicará a todo el *DataFrame* o a una columna
 - `map` : se aplicará solo a una columna



Limpieza-II-Strings.ipynb 148KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Resumen Limpieza II - Strings

Repaso Limpieza 2

- Crear nuevas columnas con `str.split()`. Los parámetros más importantes son:
 - `patrón`: que es por lo que queremos splitear
 - `n`: cuántas veces vamos a querer dividir nuestro *string*
 - `expand`:
 - si es True nos crea tantas columnas como divisiones se hayan efectuado
 - si es False no nos separa los datos, nos crea una columna y nos va a generar una lista con los elementos resultado de la división
 - `get`: nos sirve para elegir las columnas que nos queremos quedar:


```
# en caso de que me quiera quedar solo con la columna 1
.get(1)

# en caso de que quisiera quedarme con mas de una columna
.get([1,2])
```

- Métodos para aplicar a columnas y dataframes

método	aplica sobre	comentarios	Text	Text
map	aplica sobre una columna	necesitamos pasarle un diccionario		
applymap	sobre todo el dataframe	le tenemos que pasar una función		
apply	aplica sobre dataframe y columnas	le tenemos que pasar una función		

- Creación de intervalos:

- Para crear intervalos: `pd.cut(x , bins, right = True, labels = None, ordered = True)`
 - Atributos:
 - `x`: Columna por la que queremos dividir, debe unidimensional
 - `bins`: Número entero o lista de valores para los queremos dividir nuestros valores.
 - `labels`: lista o None Por defecto None.
 - `Lista`: Pasar la lista de las etiquetas por las que queremos que nos separe
 - `False`: Devuelve el número entero de cada uno de los bins - `ordered`: Por defecto True. indica si las etiquetas están ordenadas o no.

- Creación de ratios:

- Ratio: Es el cociente de las variables que queremos hacer el ratio
- Porcentaje: Es el cociente de las variables, multiplicadas por 100, para convertir el ratio en

cociente.

- Podemos aplicarlo con un lambda utilizando un apply
- Podemos hacerlo manualmente.

Pair Programming Limpieza II

Antes de seguir, recordamos las preguntas que nos planteamos al principio del _pair programming_** de EDA para dirigir nuestro análisis.**

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Cuál es el rango de edad que sufre la mayoría de los ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Si reducimos esto a una tabla para saber que tenemos ya limpio y que no. **Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✗ esta en formato <i>string</i> cuando debería ser <i>int</i> y en algunos casos tenemos rangos de edad
species	✗ es un jaleo! Debemos unificar los nombres reducir a las especies más importantes
country	✓ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas ✓
year	✓ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas ✓
fecha	✓ la limpiamos en el <i>pair</i> de Pandas ✓

Por lo tanto, en este ejercicio nos queda por limpiar las columnas de `age` y `species`, así que manos a la obra. Vamos a ver que podemos hacer para cada una de las columnas y os dejaremos algunas pistas . Para esto trabajaremos con el fichero generado en Limpieza I.

1. Columna de `species`: si exploramos esta columna en detalle nos podemos dar cuenta que tenemos muchos valores únicos y esto hace que sea muy difícil trabajar con esta columna. Lo que vamos a hacer es clasificar los tiburones en 5 especies diferentes, las más comunes, que incluyen el tiburón blanco (*White*), el tiburón tigre (*Tiger*), el tiburón gris (*Grey*), el tiburón limón (*Lemon*) y el tiburón toro (*Bull*). El resto de las especies las incluiremos en un único grupo que podremos llamar "Unspecified". Ahora nos podemos sentir un poco abrumadas y no saber como enfrentarnos a este reto, pero *don't worry*, os dejamos por aquí unas **pistas** para que os ayuden a entender cómo hacerlo.

- Los valores de las columnas son *strings* por lo que podremos usar `regex` para buscar palabras clave en cada celda y asignarlo a una de las categorías que hemos definido previamente.

```
# imaginemos que el valor de una celda es el siguiente
```

```
'White shark, 3.5 m'
```

```
# tendremos que buscar el patrón de regex que nos permita extraer White shark de ese s  
# Un patrón que podríamos usar es:
```

```
patron_blanco = r".*[Ww](hite|HITE).*" # esto podría ser así porque puede estar en may
```

```
# de la misma forma que hemos sacado el patron para el tiburón blanco, tendremos que s
```

- Tendremos que crearnos una función que aplicaremos sobre nuestra columna `species` para que nos devuelva una nueva columna con los valores clasificados en función de los patrones de regex que hayamos definido.

2. Columna de `age`: es una columna de tipo *string* pero debería ser de tipo *integer*. Además, en esta columna nos vamos a encontrar con algunos errores tipográficos, estos incluyen:

- Edad en formato *string*
- Edades separadas por `&`, `or`, `to`, `>`
- Edades con `?`

Vamos con algunas **pistas** para que os ayuden a entender cómo hacerlo:

- Primero tendremos que eliminar todos esos símbolos especiales que nos aparecen. De nuevo, podremos usar `regex` para extraer únicamente los números que es lo que nos interesa. Usar este regex en una función para sacar solo los números.

- Puede que os salga un error similar a este:

```
TypeError: expected string or bytes-like object
```

Para solucionar este problema, antes de nada tendrás que ejecutar este código para que no os de error:

```
df['nombre_columna'] = df['nombre_columna'].astype(str)
```

- Una vez que hayáis extraído los números, os daréis cuenta que hay celdas que tienen más de una edad. Tendréis que decidir qué hacer en esos casos. Os dejamos por aquí una posible opción usando un método de Pandas que os puede resultar super útil. El método `explode`, [aqui](#).
- Por último cambiad el tipo de la columna de *string* a *integer*.

3. Guarda el `csv` con las columnas limpias para seguir trabajando con este *dataframe* limpio.

Happy coding

Limpieza III - Valores extremos

Aprendemos como trabajar a como detectar y limpiar los outliers en nuestro dataset

Los *outliers* o valores perdidos son observaciones anormales en nuestros datos que puede afectar potencialmente nuestros análisis. Estos valores pueden hacer que nuestras conclusiones no sean las correctas y llevarnos a sacar conclusiones erróneas. Por esto, hoy aprenderemos:

- Como detectarlos dentro de mi *dataset*
- Una vez que los hayamos detectado, ¿Qué hacemos con ellos? . Podremos:
 - Eliminarlos
 - Reemplazarlos por el valor medio o la mediana de cada columna.



Limpieza-III-Outliers.ipynb 111KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Limpieza III

Antes de seguir, recordamos las preguntas que nos planteamos al principio del *pair programming* de EDA para dirigir nuestro análisis.

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Cuál es el rango de edad que sufre la mayoría de los ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Si reducimos esto a una tabla para saber que tenemos ya limpio y que no. **Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✓ esta en formato <i>string</i> cuando debería ser <i>integer</i> y en algunos casos tenemos rangos de edad
species	✓ es un jaleo! Debemos unificar los nombres reducir a las especies más importantes
country	✓ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas ✓
year	✓ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas ✓
fecha	✓ la limpiamos en el <i>pair</i> de Pandas ✓

Ya hemos limpiado todas las columnas que parecía que no tenían el tipo de dato que queríamos o que tenían demasiada información. Por lo tanto, en este ejercicio tendremos que limpiar los *outliers*.

1. ¿Sobre qué columnas podremos eliminar o reemplazar los *outliers*?
2. Identifica visualmente los *outliers*
3. Identifica sobre el *dataframe* las filas que son *outliers*
4. Reemplaza los *outliers* por el estadístico más correcto.
5. Guarda el `csv` con las columnas limpias para seguir trabajando con este *dataframe* limpio en el siguiente *pair programming* de limpieza.

Happy coding

Limpieza IV - Valores Nulos

En este jupyter veremos como procesar los valores nulos en nuestro dataset.

Cuando obtenemos un *dataset*, ya hemos visto que a menudo está "sucio". Los **valores faltantes** o **valores nulos** son los más comunes y fáciles de encontrar. Los diferentes métodos de procesamiento de valores nulos tienen un gran impacto en la extracción y obtención de conclusiones.

Los métodos de imputación consisten en estimar los valores ausentes en base a los valores válidos de otras variables y/o casos del *dataset*. La estimación se puede hacer a partir de la información del *dataset* completo o bien de algunas variables especialmente seleccionadas.

Principales procedimientos para imputar valores nulos:

- **Sustitución por la Media:** consiste en sustituir el valor nulo por la media de los valores válidos. Este procedimiento plantea inconvenientes como:
 - dificulta la estimación de la variáncia,
 - distorsiona la verdadera distribución de la variable, y
 - distorsiona la correlación entre variables dado que añade valores constantes.
- **Sustitución por constante:** consiste en sustituir los valores ausentes por constantes cuyo valor viene determinado por razones teóricas o relacionadas con la investigación previa. Presenta los mismos inconvenientes que la sustitución por la media, y solo debe ser utilizado si hay razones para suponer que es más adecuado que el método de la media.



Limpieza-IV-ValoresNulos.ipynb 50KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Limpieza IV

En este caso trabajaremos con el *dataframe* que limpiamos en el ejercicio de *pair* de Limpieza anterior, el de Limpieza III, donde limpiabamos las columnas de `species`, `age`, etc. y eliminábamos los *outliers*.

Antes de seguir, recordamos las preguntas que nos planteamos al principio del *pair programming* de EDA para dirigir nuestro análisis.

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Cuál es el rango de edad que sufre la mayoría de los ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Si reducimos esto a una tabla para saber que tenemos ya limpio y que no. **Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✓ esta en formato <i>string</i> cuando debería ser <i>integer</i> y en algunos casos tenemos rangos de edad
species	✓ es un jaleo! Debemos unificar los nombres y reducir a las especies más importantes
country	✓ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas V
year	✓ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas V
fecha	✓ la limpiamos en el <i>pair</i> de Pandas V

Es el momento de ponernos a trabajar con los valores nulos . A lo largo de este ejercicio de *pair programming* vamos a intentar eliminar los valores nulos de nuestras columnas. En la lección hemos aprendido varios métodos, nosotras os planteamos los ejercicios pero sentiros libres de usar el método que más se adapte a vuestras necesidades. Manos a la obra!

1. Lo primero que tenemos que evaluar es en que columnas tenemos nulos y que cantidad tenemos en cada una. ¿Hay alguna columna con una gran cantidad de nulos? En caso de que sea así deberemos eliminarla.
2. Es el momento de eliminar los nulos:
 - Reemplazad los valores nulos de la columna `age` por la media de la edad, redondeada a dos decimales.
 - En relación a la columna de `country` al tratarse de una columna de tipo categórica, reemplazad los valores nulos por una nueva categoría que se llame `Unknown` .
 - Reemplazad los valores nulos de la columna `fatal` por `Unknown` .
 - Reemplazad los valores nulos de la columna `type` por el valor más frecuente (la moda).
 - Reemplazad los valores nulos de la columna `fecha` por `Unknown` .
3. Guardad el `csv` para seguir trabajando con él en los siguientes ejercicios de *pair*.

Happy coding

Limpieza V - Valores Nulos Sklearn

En este jupyter veremos como procesar los valores nulos en nuestro dataset usando la librería `sklearn`

Estos métodos incluyen:

- *Simple Imputer*: reemplazar los valores nulos por uno especificado por nosotros.
- *Iterative Imputer*: nos reemplazará los valores nulos en función de otras variables de nuestro set de datos.
- *KNN Imputer*: nos permite reemplazar los nulos en función de los algoritmos de vecinos más cercanos.

Sin embargo, estos métodos también tienen sus inconvenientes:

- Incrementan artificialmente las relaciones entre variables.
- Asumen que las variables con valores nulos tienen relación de alta magnitud con las otras variables.



Limpieza-V-ValoresNulosSklearn.ipynb 30KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Limpieza V

En este caso trabajaremos con el `dataframe` que limpiamos en el ejercicio de *pair* de Limpieza III donde limpiabamos las columnas de `species`, `age`, etc. y eliminábamos los *outliers*. Hoy volveremos a gestionar valores nulos, pero en este caso usaremos los métodos de imputación de `sklearn`.

Antes de seguir, recordamos las preguntas que nos planteamos al principio del *pair programming* de EDA para dirigir nuestro análisis.

- ¿Es Australia es el sitio más peligroso y letal para estar relajada en la playa?
- ¿Cuál es el rango de edad que sufre la mayoría de los ataques?
- Independientemente de la edad, sufren los hombres más ataques que las mujeres?
- ¿En qué mes ocurren más ataques?
- ¿Cuál es la relación entre la especie y el tipo de ataque (si es fatal o no)?
- ¿Cómo han evolucionado los ataques a lo largo del tiempo?

De todo esto, nos damos cuenta que solo tenemos que limpiar algunas de las columnas, en concreto `age`, `species`, `country`, `fatal`, `year`, `sex`. Si reducimos esto a una tabla para saber que tenemos ya limpio y que no. **Actualizamos esta tabla ya que en el ejercicio de *pair* de Limpieza I ya dejamos algunas columnas limpias:

variable	¿Está limpia?
age	✓ esta en formato <i>string</i> cuando debería ser <i>integer</i> y en algunos casos tenemos rangos de edad
species	✓ es un jaleo! Debemos unificar los nombres reducir a las especies más importantes
country	✓ los países están en mayúsculas, algunos se repiten con algunos cambios
fatal	✓ la limpiamos en el <i>pair</i> de Pandas V
year	✓ es una columna de tipo <i>float</i> deberíamos convertirla a <i>integer</i>
sex	✓ la limpiamos en el <i>pair</i> de Pandas V
fecha	✓ la limpiamos en el <i>pair</i> de Pandas V

Es el momento de ponernos a trabajar con los valores nulos de nuevo . A lo largo de este ejercicio de *pair programming* vamos a intentar eliminar los valores nulos de nuestras columnas. En la lección hemos aprendido varios métodos de skelarn intentemos aplicarlos todos. Manos a la obra!

1. Es el momento de eliminar los nulos:

- Reemplazad los valores nulos de la columna `age` por la media de la edad usando el método `SimpleImputer` .
- Reemplazad los valores nulos de la columna `sex` por la moda, usando el método `SimpleImputer` .
Pista La moda en este tipo de aproximación se indica como `most_frequent` .
- Reemplazad los valores nulos de la columna `type` por el valor más frecuente (la moda) con el método `SimpleImputer` .
- Utilizad el método `KNN Imputer` para reemplazar todos los valores nulos de las columnas numéricas.
- Utilizad el método `Iterative Imputer` para reemplazar todos los valores nulos de las columnas numéricas.
- ¿Podrías explicar qué diferencia hay entre estos dos últimos métodos?

2. Guardad el `csv` para seguir trabajando con el en los siguientes ejercicios de *pair*

Happy coding

ETL

Datos-ETL

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de ETL

En esta apartado encontrareis los ficheros que necesitaréis para las lecciones de EDA:

- **Sesión teórica**

- [datos_Madrid.csv](#)



[datos_Madrid.csv](#) 6KB

Binary

[Descarga este fichero.](#)



[datos_actualizados.csv](#) 11KB

Binary

[Descarga este fichero.](#)

- **Sesión práctica**

- [attacks_limpieza_completa.csv](#)



[attacks_limpieza_completa.csv](#) 703KB

Binary

[Descarga este fichero para poder hacer los ejercicios que os planteamos.](#)

- [datos_union_clima_ataque.csv](#)



[datos_union_clima_ataque.csv](#) 2MB

Binary

[Descarga este fichero para poder hacer los ejercicios que os planteamos.](#)

ETL I - Extracción - API's

¿Qué significan las siglas ETL? Aprenderemos sobre las operaciones de tipo Extracción (Extract) - Carga (Load) - Transformación (Transform) para el procesado de los datos

Vamos a aprender sobre las ETL y su utilidad, con la finalidad de ser capaces de desarrollar y automatizar este tipo de procesos para simplificar el proceso de la transformacion de los datos en crudo a datos refinados. Vamos a ello!



ETL-1.ipynb 47KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso Introducción a las ETL

- Fases de una ETL:

- Extract : extraer los datos de bases de datos, textos, email, páginas web, etc.
- Transform : procesado de la información, haciendo la limpieza de los datos
- Load : lo cargamos en csv, json, BBDD.

- Es importante la respuesta de la API.

- **200**: sería que la llamada a la API fue exitosa
- **403**: prohibido.
[códigos](#)

- Extract : tipos de archivos que podemos usar:

- xls
- csv
- los resultados de una API
- BBDD, query
- API's

- Si queremos trabajar con **API's**

- librería `requests` : nos permite acceder al contenido de la API.
- Pasos:

```
# la llamada a la API
response = requests.get(url)

# para ver si la llamada se hizo con éxito
response.status_code
200, 403, 203

# para ver lo que nos devuelve la API.
response.text

response.json()

# convertir a dataframe

df = pd.json_normalize(response.json())
```

Pair Programming ETL I

Vamos a nutrir los datos de los ataques de los tiburones con información climática de los países que tenemos. Para eso vamos a usar la API del clima que hemos aprendido en la clase invertida:

```
url = f'http://www.7timer.info/bin/api.pl?lon={lon}&lat={lat}&product={producto}&output=json'
```

En este caso os recomendamos que uséis el producto `meteo` para obtener la información climática. Para hacer la llamada a la API necesitamos también las coordenadas de los países que tenemos en el *dataset*.

Por lo tanto, el objetivo es que saquéis la información del clima de la API para los países que tenemos. Pero antes de poneros manos a la obra, tenemos muchísimos países y esto puede hacerse eterno. Sacad la información climática solo para los siguientes países, con las siguientes coordenadas:

Pais	Latitud	Longitud
USA	39.7837304	-100.445882
Australia	-24.7761086	134.755
South Africa	-28.8166236	24.991639
New Zealand	-41.5000831	172.8344077
Papua New Guinea	-5.6816069	144.2489081

Requisitos de este ejercicio de *pair programming*:

- Deberéis meter toda la información climática en un único *dataframe*.
- Deberéis hacer la llamada a la API de una sola vez. Es decir, tendréis que iterar por la lista de países y sacar la información del clima para cada uno de ellos.
- Al meter toda la información en un único *dataframe* tendremos que crear una columna que corresponda con el nombre del país.
- Guardar el *dataframe* obtenido para usarlo en la siguiente sesión de *pair programming*.

Pistas :

- Crearos un diccionario donde:
 - Las `keys` sean los nombres de los paises
 - Los `values` sean las coordenadas de los paises en formato lista.
- Iterar por el diccionario. Dentro del `for` haremos la llamada a la API como hemos aprendido en las clases invertidas. Recordamos que para iterar por el diccionario tenemos que seguir la siguiente estructura:

```
for key, value in diccionario.items():
    print(key, value)
```

- Crearnos un *dataframe* vacío donde iremos concatenando los datos obtenidos de la API cada país.
- Guardad el csv ya que mañana lo necesitaremos para hacer el ejercicio de *pair programming*.

Happy coding

ETL II - Transformación I - Limpieza

En los procesos ETL, tras la extracción de los datos llega la segunda fase: la transformación. Esta fase consiste en la limpieza y enriquecimiento de los datos para convertirlos en datos.

Para entender la necesidad de un proceso de transformación debemos tener en cuenta que en un proceso ETL se manejan fuentes diversas, como puede ser información de una web, cualquier tipo de descarga de Internet, una API, etc.

Esta variedad de fuentes de datos, con diferentes características, imposibilita o dificulta la posibilidad de realizar comparaciones si con anterioridad no se realizan conversiones y formateos. De ahí la necesidad de los procesos de transformación.

Las acciones o procesos más habituales son:

- Reformateo de datos.
- Conversión de unidades. Por ejemplo, convertir millas en kilómetros por hora o viceversa.
- Selección de columnas para su carga posterior. Por ejemplo, hacer que las columnas con valores nulos no se carguen.
- Agregación de columnas. Añadir una columna con la procedencia de determinados automóviles sería un ejemplo.
- Dividir una columna en varias. Esta acción resulta de gran utilidad para, por ejemplo, separar en tres columnas, una para el nombre y otras dos para los apellidos, la identificación de una persona que antes estaba en un solo campo.
- Obtener nuevos valores calculados.
- Unir datos de varias fuentes. O actualizar los que tenemos.



ETL-2.1.ipynb 71KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming ETL Transformación I

Tendréis que usar el csv `attacks_limpieza_completa` que tenéis adjunto abajo.

En la lección de hoy aprendimos como transformar nuestros datos para que estén preparados para almacenarlos en una BBDD. En este momento tenemos dos fuentes de datos:

1. El csv con los ataques de tiburones que hemos estado limpiando hasta ahora, el que os hemos adjuntado (`attacks_limpieza_completa`). Sintiros libres de usar vuestros propios csv en caso de que queráis.
2. El csv con los datos climáticos de los principales países que tienen ataques de tiburones, el que creamos en el *pair programming* de ayer.

El objetivo de la sesión de hoy será juntar en un único csv la información de ambas fuentes. Para ello:

- Cargaremos los dos ficheros de datos
- Del *dataframe* de los ataques nos quedaremos solo con las filas de los países que seleccionamos en la lección de ayer:
 - USA
 - Australia
 - New Zealand
 - South Africa
 - Papua New Guinea
- Del *dataframe* de los datos climáticos seleccionaremos todas las columnas.
- Cuando ya tengamos todos los datos deseados juntaremos los dos csv.
 - Para hacer esta unión tendremos que hacer un *groupby* en la tabla de clima para sacar una media de las medidas climáticas por país.
 - Antes de hacer el *groupby* si nos fijamos tenemos dos columnas `rh_profile` y `wind_profile` cuya información es una lista de diccionarios. Si intentamos hacer la media de eso no nos dará un valor real. A este problema ya nos enfrentamos en la clase invertida de ETL-2, donde teníais un Bonus para desempaquetar esta información. En caso de que en aquel ejercicio no lo consigierais os dejamos por aquí una posible solución que nos permite separar esa información en distintas columnas. Os dejamos el código documentado. ▲ Os recomendamos que vayáis desgranando el código y viendo lo que nos devuelve cada línea de código para entenderlo mejor.

```
# os recomendamos resetear el index del dataframe de los datos climáticos para que no :
# El primer problema al que nos podemos enfrentar es que si vemos los tipos de las col
clima.dtypes

timepoint          int64
cloudcover         int64
highcloud          int64
midcloud           int64
lowcloud           int64
rh_profile         object
wind_profile       object

# en Python tenemos la librería `ast` que nos permite castear un string que dentro tie
import ast

clima['wind_profile']= clima['wind_profile'].apply(ast.literal_eval)

# una vez que tengamos la columna cambiada, una fantasía de Pandas es que si hago un a
x = clima['wind_profile'].apply(pd.Series)
```

```
# nos creamos un dataframe nuevo con el resultado de la información de una de las columnas
x = df['rh_profile'].apply(pd.Series)

# ¿Qué es lo que ocurre cuando hacemos esto?
# Nos ha creado tantas columnas como valores tuvieramos en la lista. Donde columna es, es el nombre de la lista

# Ok, hemos conseguido desempaquetar la información de la lista en distintas columnas.

# Por eso empezamos con un for para iterar por cada una de las columnas.
for i in range(len(x.columns)):

    # aplicamos el apply, extraemos el valor de la key "layer" y lo almacenamos en una variable
    nombre = "rh_" + str(x[i].apply(pd.Series)[["layer"]][0])

    # hacemos lo mismo con una variable que se llame valores para "guardar" los valores
    valores = list(x[i].apply(pd.Series)[["rh"]] )

    # usamos el método insert de los dataframes para ir añadiendo esta información a el
    df.insert(i, nombre, valores)

# una vez que hayamos hecho esto para las dos columnas ya podremos hacer el groupby para agruparlos
```

- Guardar los resultados obtenidos en un csv que usaremos en próximos ejercicios de *pair programming*.

Happy coding

ETL III - Transformación II - Clases y Funciones de limpieza

En los procesos ETL, tras la extracción de los datos llega la segunda fase: la transformación. Esta fase consiste en la limpieza y enriquecimiento de los datos para convertirlos en datos.

Esta lección es continuación de la lección anterior. ¿Recordáis cuando vimos las Clases en el primer módulo? Pues el momento de volver sobre ellas, el objetivo de este jupyter, meter todo el código que teníamos en una sola clase.



ETL-2.2.ipynb 46KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming ETL Transformación II

En la lección de hoy aprendimos como Crearnos una clase que nos permita limpiar los datos obtenidos de la API.

En este ejercicio, tendréis que crear una clase con el código que usamos en los ejercicios de *pair programming* de ETL Transformación I y II.

Happy coding

ETL IV - Carga I - Creación BBDD e inserción de datos

El último paso de una ETL es cargar los datos obtenidos en una BBDD de SQL, la nube etc. En nuestro caso lo haremos en SQL.

Esta es la parte final de nuestro proceso ETL. Se trata de migrar los datos al destino final. Puede ser un almacén de datos o una base de datos local o en la nube. Estos datos pueden actualizarse automáticamente cuando se extraen y transforman nuevos datos. Estos datos ordenados y organizados son utilizados posteriormente por los analistas de negocio para su visualización y exploración, por los científicos de datos para la experimentación y predicción, o por otros usuarios finales.



ETL-3.1.ipynb 51KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming ETL Carga I

Es el momento de meter todos nuestros datos en SQL !!! En este ejercicio nos crearemos dos tablas en una BBDD creada por nosotras. Una de las tablas contendrá la información que obtuvimos de los ejercicios de *pair programming* de Limpieza, es decir, el *data set* de ataques de tiburones limpido. La segunda tabla tendrá la información obtenida en el ejercicio de *pair* de ETL 1.

Nota Todo lo tendremos que hacer desde *jupyter notebook*

1. Cread la BBDD con el nombre de `tiburones` .

2. Cread las tablas de la BBDD:

- Tabla `ataques`
- Tabla `clima`

3. **BONUS** Insertar los datos en las tablas.

Nota Esta parte del *pair* es optativa y no será considerada para la evaluación

En caso de que no tengáis los datos unidos de la sesión anterior, tenéis un csv `datos_union_clima_ataques` con todos los datos que necesitareis para este ejercicio



[soporte.py](#) 6KB

Binary

Descarga este archivo en VS Code para seguir la lección.

ETL V - Carga II - Clases y Funciones BBDD e inseción

El último paso de una ETL es cargar los datos obtenidos en una BBDD de SQL, la nube etc. En nuestro caso lo haremos en SQL.

Igual que en la lección de transformación, el objetivo de esta lección es meter todo el código en un Clase para intentar automatizar nuestro código al máximo!



ETL-3.2.ipynb 24KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming ETL Carga I

Igual que en el ejercicio de *pair programming* de ETL Transformación II, tendréis que crear una clase que nos permita cargar los datos en SQL I.

Happy coding

ETL VI- Ejecutable - Pipeline ETL

En la lección de hoy aprenderemos como crear un ejecutable en Python. El objetivo, juntar todo el código creado hasta ahora en un único fichero que nos permita interactuar y se ejecute todo de una vez

Hasta hoy hemos aprendido como crear cada uno de los pasos de una ETL por separado. Han sido lecciones intensas donde hemos aglomerado muchos de los conocimientos adquiridos hasta hoy. Hoy será sencillo, lo único que haremos será unificar toda la información en un único fichero, un `.py` que será el ejecutable. Un ejecutable es un fichero que se puede ejecutar de forma independiente desde la terminal y que nos permitirá no tener que ir ejecutando línea a línea el código.



ETL-4.ipynb 5KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



main.py 4KB

Binary

Descarga este archivo en VS Code para seguir la lección.



soporte.py 6KB

Binary

Descarga este archivo en VS Code para seguir la lección.

Pair Programming ETL IV

En la lección de hoy aprendimos como podemos crear un ejecutable. En este contexto, el objetivo del ejercicio de hoy es que creeis un ejecutable con todo el código que hemos ido creando durante las otras sesiones de *pair programming* ETL.

Al final, debereis tener un archivo `.py` que lo ejecutemos en la terminal y nos actualice los datos, los limpie y nos los meta en SQL.

Happy coding

Repasos

Repaso NumPy y Pandas I

Vamos a aplicar los conceptos de hemos aprendido de NumPy y Pandas

NumPy Repaso

En este jupyter se os plantean una serie de preguntas relacionadas con NumPy que deberéis contestar.



repaso_numpy_v2.ipynb 5KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



repaso_numpy_v2_Soluciones.ipynb 26KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Un extra de tratamiento de imágenes con numpy con algunos métodos nuevos que no vimos en las clases:



repaso_numpy_extra_imagenes.ipynb 1MB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pandas repaso

Realizaremos un análisis exploratorio y contestamos a la siguiente pregunta:

¿Es la colonia de Bustamante más cara que la de Paula?

Los datos para este repaso los encontraréis en [este](#) link.



repaso_pandas_I.ipynb 4KB

Binary

Descarga este Jupyter y ábrelo en VS Code.



repaso_pandas_I_Soluciones.ipynb 70KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Matriz de la exploración

- Columnas del dataframe

```
df.columns
```

- Forma del dataframe

```
df.shape
```

- Tipos de datos, nombres de las columnas, los elementos no nulos

```
df.info()
```

- Para ver solo los tipos de los datos

```
df.dtypes
```

- Principales estadísticos de las columnas

```
# para las variables numéricas  
df.describe().T # el .T nos transpone el resultado del describe
```

```
# para las variables categóricas  
df.describe(include="object")
```

- Los duplicados

```
df.duplicated().sum()
```

- Los valores nulos

```
df.isnull().sum()  
df.isna().sum()
```

NOTA También tenemos el `notnull` que nos devuelve lo que no son nulos.

- Para las variables categóricas o de tipo string sacar las frecuencias de cada una de las categorías

```
# los valores únicos y sus frecuencias  
df["col"].value_counts()
```

```
# para sacar solo los valores únicos  
df["col"].unique()
```

- Programar y analizar los resultados.

COSAS IMPORTANTES

- Para Pandas, los NaN son de tipo float.
- Menos es más, si tenemos una solución sencilla la usamos.
- Convertir una columna a tipo datetime con `pd.datetime`.
- Transponer el describe nos puede ayudar a entender mejor los resultados.

Minimatría de limpieza

- Tipo de datos si son correctos
- Los valores de los *strings* y hacer limpieza sobre ellos (convertir todo a minusculas, quitar espacios, quedarnos con la información que nos interesa, etc.)
- Gestionar los nulos
- Gestionar/Ver los valores extremos *outliers*.
- Unificar el nombre de las columnas, todas en minúsculas, sin espacios

Repaso II- Pandas Visualización Y Estadística

Vamos a aplicar los conceptos de hemos aprendido de NumPy y Pandas

En este repaso vamos a trabajar en el siguiente contexto.

Trabajamos como analistas de datos para el supermercado día, nuestro jefe nos ha compartido dos ficheros que podremos encontrar en [este link](#). Nos encontraremos con csv:

1. `precios_productos` : en el que podremos encontrar una serie de productos, y sus precios
2. `productos_dia` : en el que encontramos todos los productos que han comprando y vendido en el supermercado desde el 2021.

En concreto nuestro jefe nos ha pedido que:

- Juntemos los dos csv en uno único.
- Identifiquemos la categoría y subcategoría de cada producto.
- Hagamos un análisis temporal de la evolución de los precios de los productos.
- Identifiquemos si existe alguna relación entre el `price` y el `reference_price`
- Hagamos un análisis de las categorías:
 - ¿Qué categoría es la que vende más productos?
 - Dentro de cada categoría, ¿qué subcategorías son más populares?
 - ¿Qué categoría es la que tiene los precios más altos? ¿y los mínimos? ¿Cuál es su media?



[repaso_viz_pandas_esta_soluciones.ipynb](#) 1MB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso III- EDA - Limpieza

Trabajamos para una inmobiliaria, nuestros jefes han obtenido una serie de datos sobre los inmuebles que están en alquiler en toda España. Ellos están iniciando su negocio y quieren focalizarse en aquellas comunidades autónomas donde más movimiento de compra-venta de inmuebles. Ojo!! Solo les interesa los inmuebles para vivir, piso, chalet, etc. En concreto, quieren identificar las 5 CCAA que tienen más inmuebles en alquiler y contestar a una serie de preguntas

- ¿Cuál es el precio medio por tipo de inmueble en función de la provincia?
- ¿Qué tipo de inmuebles son los que más se ofertan?



[casas2.csv](#) 5MB

Binary

[Descarga este csv .](#)



[repaso-EDA-Limpieza_Promo-C.ipynb](#) 499KB

Binary

[Descarga este jupyter para ver lo que vimos en la clase .](#)

Descripcion del conjunto de datos

- HY_id: identificación del inmueble en Haya Real Estate
- HY_tipo: clase de inmueble (piso, garaje, casa de pueblo...)
- HY_cod_postal: código postal del inmueble
- HY_provincia: provincia donde está situado el inmueble
- HY_descripcion: descripción del inmueble
- HY_distribucion: distribución del inmueble
- HY_antiguedad: año de antigüedad del inmueble
- HY_metros_utiles: metros cuadrados útiles del inmueble
- HY_metros_totales: metros cuadrados totales del inmueble
- HY_num_banos: número de baños
- HY_cert_energ: calificación del certificado energético
- HY_num_terrazas: número de terrazas
- HY_ascensor: existencia o no de ascensor
- HY_trastero: existencia o no de trastero
- HY_num_garajes: número de garajes
- HY_precio: precio de venta del inmueble
- HY_precio_anterior: precio anterior
- IDEA_area: Superficie de la zona
- IDEA_poblacion: Población estimada de la zona
- IDEA_densidad: Densidad de población de la zona

Repasso IV- API's

Vamos a aprender a usar la API de Spotify

En este jupyter aprenderemos como sacar información de la API y como enfrentarnos a un json con mucha información



Spotify.ipynb 1MB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Proyecto 2

Se va a tomar el dataset que se ha utilizado para el proyecto del módulo 1, una vez aplicada la limpieza preliminar. Con estos archivos procesados vamos a montar de nuevo el dataset.

Índice

- Resumen
- Objetivos
- Caso de uso
- Especificaciones
- Planificación del proyecto
 - *Sprints*
 - Historias de usuario
- Entrega
- Presentación

Resumen

En este proyecto vamos a aprender a tratar con los archivos que se han obtenido de el procesamiento de los ficheros del proyecto del módulo 1. Para ello tenemos que aprender a unificar diferentes fuentes de datos en un único fichero, aplicar la limpieza que nos parezca conveniente, transformar los datos y ser capaz de extraer conocimiento de la información contenida. Para ello vamos a tener que presentar un informe al final del módulo con los resultados obtenidos de forma visual y ser capaces de explicar algunas cosas interesantes del análisis realizado ¡Esta será vuestra segunda experiencia de trabajo en equipo relacionada con programación! ¿Estáis preparadas?

Objetivos

1. Consolidar los conocimientos de Numpy, Pandas, Matplotlib y Seaborn. Para procesar y unificar diferentes ficheros de datos en un mismo formato. Consolidar los conocimientos de Python básicos, así como el tratamiento de ficheros en diversos formatos y extracción de datos de una base de datos.
2. Realizar un análisis exploratorio de datos exhaustivo, con el fin de entender los datos de entrada y ser capaces de identificar que datos necesitan ser limpiados mediante técnicas de visualización.
3. Implementar *Scrum* como marco de referencia para el desarrollo del producto, basándonos siempre en los valores de *Agile* como puntos clave del trabajo en equipo y la mejora continua.
4. Mejorar la comunicación entre los miembros del equipo.
5. Mejorar vuestras habilidades de comunicación en público al exponer el proyecto en la sesión final.

Caso de uso

Con este proyecto vais a demostrar que sois capaces de unificar un conjunto de datos bajo un mismo archivo, realizar un análisis exploratorio de datos y visualizaciones explicativas. Esto os permitirá mostrar vuestras habilidades a enfrentarnos a un conjunto de datos haciendo uso de las librerías más comunes para el tratamiento de datos en Python, algo que os será útil a la hora de demostrar vuestros conocimientos a las empresas durante los futuros procesos de selección a los que os enfrentais.

Especificaciones

En desarrollo del procesado del conjunto de datos haremos uso de las siguientes tecnologías y paquetes de Python:

- **Pandas**: Unión de fuentes de datos, procesado y limpieza de datos.
- **Matplotlib/Seaborn**: Para la visualización de datos.
- **Sidetable**: Para obtener estadísticas de los conjuntos de datos.
- **Sckit-Learn**: Para codificar los datos y poder normalizarlos.

El proyecto deberá contener los siguientes elementos:

1. Un repositorio con los ficheros utilizados, así como el código empleado para el desarrollo del proyecto. Siguiendo una estructura de carpetas coherente y sencilla.
2. Habrá que asegurarse que todos los archivos finales sean entregados.
3. Elaboración de una presentación para el día de la demo.

Planificación del proyecto

Sprints

Para la realización de este proyecto trabajaremos en 3 *sprints* (3 iteraciones) de entre 5-7 sesiones cada iteración. Siguiendo los principios ágiles, estableceremos pequeños ciclos iterativos de forma que al final de cada uno generemos valor perceptible por nuestros usuarios. Dedicaremos el primer día a la planificación del *sprint* (**sprint planning**) y el resto a trabajar en el desarrollo del proyecto. Al final de cada *sprint* haremos un *Sprint Review* del proyecto para presentar los resultados conseguidos y recoger *feedback*, al igual que una retrospectiva (retro) para evaluar cómo ha ido el *sprint*, además de valorar vuestro trabajo en equipo de cara a mejorar en el siguiente *sprint*.

También haremos una retro corta revisando los *working agreements* que hemos acordado al inicio del proyecto y añadiendo cualquier otro *feedback* que nos permita mejorar el proyecto.

Al final del *sprint* (que coincidirá con el final del proyecto), haremos una sesión de presentación más completa, más allá de lo que sería un *Sprint Review*

Historias de usuario

Para la gestión del proyecto, usaremos historias de usuario, que es una herramienta para definir las características de un producto que veremos en detalle durante el curso.

1. Unión automatizada de los diferentes ficheros de entrada.

Valor

El cliente quiere tener todos los datos en una misma fuente, ya que únicamente va a poder trabajar con un único tipo de datos de ahora en adelante.

Contexto

El cliente ha tenido una reunión con el equipo de datos de la empresa y le han comunicado que debido a problemas internos y mejoras de infraestructura durante el próximo año únicamente van a poder dar soporte a una tipo de datos, por lo tanto van a tener que de forma temporal unificar los diferentes datos disponibles, en lo que mejoran los sistemas.

Criterios de aceptación:

- Crear la infraestructura necesaria: repositorio en GitHub y con acceso para todos los miembros del equipo.
- Unir los diferentes ficheros de entrada en uno que agregue toda la información.
- Unir los diferentes ficheros de entrada en uno que agregue toda la información y guardarlo en csv.
- **DOD: Tener en el repositorio de GitHub el archivo unificado en extension csv.**

2. Limpieza de los datos.

Valor

El cliente quiere tener todos los datos unificados con nombres más descriptivos, hacer una selección de las variables más interesantes y realizar un estudio de las características de los datos.

Contexto

En este caso el cliente, no necesita todas las variables que se tienen en el conjunto de datos, ya que se espera poder presentar un informe en el futuro, nos ha pedido que seleccionemos aquellas variables que puedan resultar más interesantes para su análisis.

Criterios de aceptación

- Realizar una selección de las variables que resulten más interesantes de análisis.
- Realizar la limpieza de las columnas seleccionadas.
- Guardar el dataset limpio en un fichero csv diferente al original.
- **DOD: Tener otro conjunto de datos con las columnas que posteriormente realizaremos en análisis en formato csv y en el repositorio de GitHub.**

3. Análisis exploratorio de datos

Valor

El cliente quiere saber qué tipo de información contiene las variables seleccionadas, ya que desea entender en profundidad las peculiaridades de los datos disponibles.

Contexto

Se desea conocer en mayor detalle sobre los datos seleccionados para el análisis, por tanto, le gustaría que se realizase un análisis exploratorio de datos con el fin de identificar datos extraños que deban ser limpiados, antes de proceder a la creación de gráficas explicativas.

Criterios de aceptación

- Obtener algunas estadísticas e información sobre la naturaleza de los datos y guardarlos en un csv o excel.
- Explicar los resultados.
- **DOD: Extraer los datos de las estadísticas y guardar en fichero externo, las soluciones en el repositorio de GitHub.**

4. Obtención de gráficos

Valor

El cliente quiere obtener conocimiento de la información contenida en los datos que han sido seleccionados para el análisis, para poder dar explicaciones relevantes sobre los datos disponibles.

Contexto

Se desea conocer en mayor detalle sobre los datos seleccionados para el análisis, debido a esto y con vista a la presentación de un informe nos ha pedido que realicemos algunas gráficas con las que posteriormente se pueda explicar algunos detalles relevantes sobre los datos disponibles.

Criterios de aceptación

- Tener como mínimo tres gráficas ilustrativas, se recomienda algún histograma.
- **DOD: Gráficas guardadas en algún ficheros de tipo png,jpeg o jpg. Así como que esten subidas al repositorio de GitHub.**

Ficheros

Los ficheros a descargar son los siguientes:



exported_db_data.csv 1MB
Binary

[Descarga este fichero.](#)



data_txt_clean.txt 14MB
Binary

[Descarga este fichero.](#)



data_xml_clean.xml 3MB
Binary

[Descarga este fichero.](#)



data_remaning_kaggle.csv 16MB
Binary

[Descarga este fichero.](#)



kaggle_survey_2021_answer_choices.pdf 158KB
PDF

[Descarga este fichero.](#) Para entender las columnas y sus respuestas.

Entrega

El formato de entrega de este proyecto será mediante la subida de este a la plataforma de GitHub. Para subirlo, se creará un repositorio en la organización de Adalab. El nombre del repositorio deberá estar compuesto de las siguientes partes, todo ello separado por guiones:

- La palabra project-da.
- Letra de la promoción promo-B.
- Número del módulo module-2.
- Número del equipo team-X. Por ejemplo:
 - Adalab/project-da-promo-b-module-2-team-1
 - Adalab/project-da-promo-b-module-2-team-3

En lo relacionado en las fechas de los *sprints*:

- Entrega del primer *sprint* (*sprint review*): 16 Diciembre.
- Entrega del segundo *sprint* : 3 Enero
- Entrega del tercer *sprint* (*sprint review*): 13 Enero.
- Demo del proyecto (presentación final): 17 Enero.

En las *sprint review* se revisará que se hayan solucionado todas las tareas técnicas asociadas a la entrega de esos *sprints*, si algo quedara pendiente se arrastraría al siguiente *sprint*.

Presentación

El último día del módulo presentaréis la versión final de este proyecto a vuestras compañeras y al equipo de Adalab. Cada equipo realizará una presentación de 5 minutos y posteriormente habrá 5 minutos de *feedback* por parte del público. En este caso, la audiencia podría ser más variada pues no sólo estarán los profesores.

El objetivo es que practiquéis la realización de las demos de los proyectos que habéis desarrollado, explicándolo desde un punto de vista técnica y también desde la perspectiva del producto, mejorando además vuestras habilidades de exposición, objetivo de desarrollo profesional del curso.

Para que la presentación salga bien es imprescindible una buena preparación. Por ello, durante el segundo *sprint* del módulo tendréis que asignar responsabilidades dentro del equipo relacionadas con la preparación de ésta. A continuación incluimos algunos elementos que os pueden ayudar a enfocar la presentación:

- En el público habrá personas con conocimientos técnicos y no técnicos.
- La parte central de la presentación será mostrar el software desarrollado funcionando, a ser posible en directo de forma dinámica o a través de un vídeo (si no fuera posible, como plan B).
- En este módulo, de los diferentes elementos adicionales que os proponemos, sería útil incluir una breve presentación de los diferentes integrantes del equipo desde un punto de vista profesional. Se trata de practicar vuestro "relato" profesional en versión muy corta. Que las personas asistentes conozcan quienes sois como profesionales. Os será también útil para las entrevistas de trabajo.
- Todas las participantes del equipo deben hablar en la presentación (sin práctica no hay mejora).

Además de esto, para mejorar vuestras habilidades de exposición en público y hacer la presentación más rica, podréis incorporar otros elementos adicionales (son solo ideas, sentíos libres de innovar y ser creativas):

- Dejar muy claro quién ha sido vuestro cliente y qué fue lo que os pidió.
- Explicar qué necesidades cubre o qué problemas soluciona el producto, cuál es el beneficio principal que aporta y qué lo hace único comparado con otros productos parecidos del mercado.
- Aportaciones "únicas y diferenciadoras" de cada equipo al proyecto.
- Cómo ha sido la organización del equipo, el reparto de tareas y la coordinación a la hora de trabajar todas en el mismo código.
- Cuál de las tareas o los puntos ha sido el que más esfuerzo ha requerido.
- Cuál de las tareas o partes del proyecto es la que hace que el equipo esté más orgulloso.
- Las tecnologías qué habéis utilizado y para qué sirven, y algunas partes del código que habéis desarrollado que merezca la pena resaltar.
- La presentación debe tener un "buen inicio y un buen cierre" que nos haga a todos estar atentos y aplaudir... ahí os dejamos que echéis a volar vuestra imaginación.
- No habléis en primera persona de lo que habéis hecho, hablad del equipo.
- No mencionéis problemas, sino "retos" que os han hecho aprender y crecer.

MODULO 3: Machine Learning y Dashboards

Regresión Lineal

Datos-Regresión-Lineal

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de ETL

En esta apartado encontrareis los ficheros que necesitaréis para las lecciones de EDA:

- **Sesión teórica**

- `boston.csv`



`boston.csv` 38KB

Binary

Descarga este fichero para poder hacer los ejericicios que os planteamos.

- **Sesión teórica ejercicios clase invertida**

- `cars.csv`



`cars.csv` 2KB

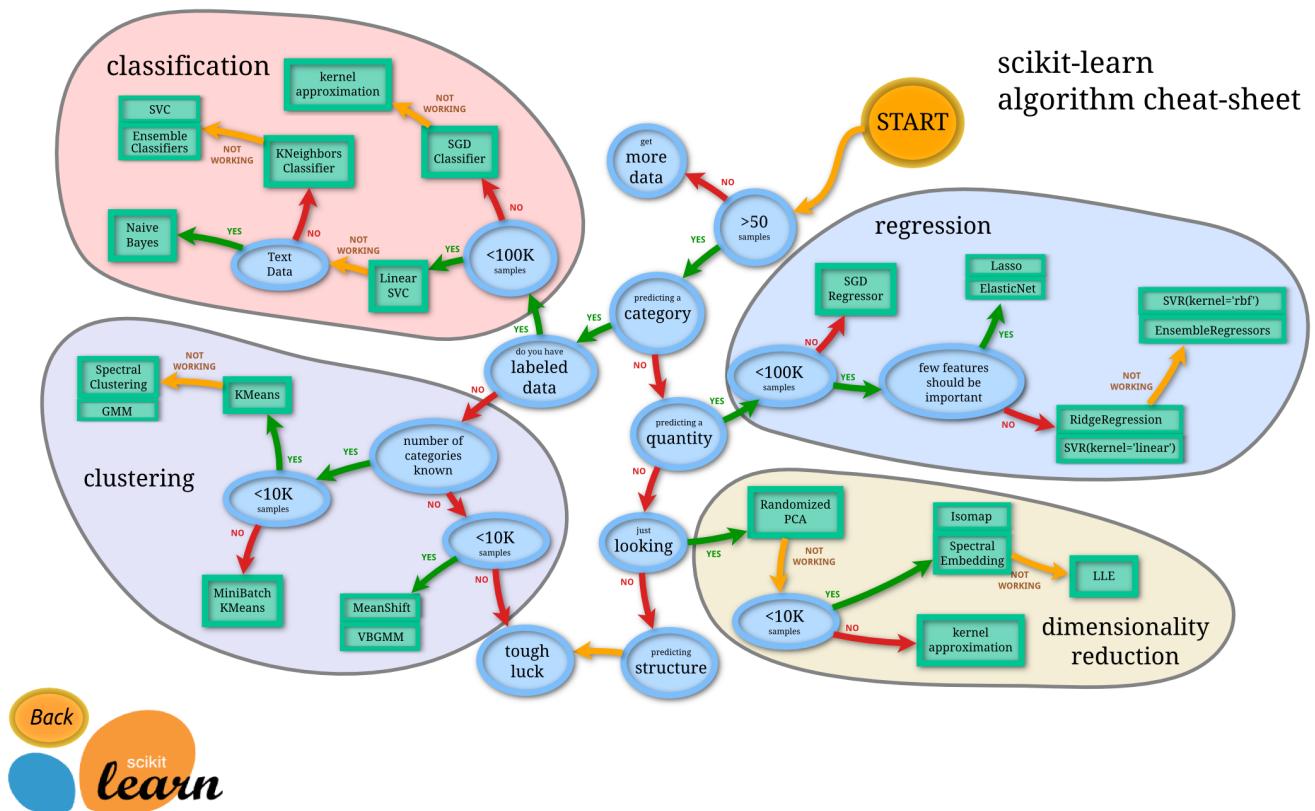
Binary

Descarga este fichero para poder hacer los ejericicios que os planteamos.

Intro a Machine Learning

¿Qué es el Machine Learning? En esta lección haremos una primera introducción al Machine Learning y haremos un pequeño recordatorio de lo que es un análisis exploratorio.

Cheat sheet ML



Cheat Sheet ML Sklearn



Regresion-Lineal-I-Machine-Learning-Intro.ipynb 1MB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Intro Machine Learning

Empezamos una nueva lección, y este es el momento de poner en práctica los conocimientos adquiridos sobre la *Regresión Lineal Simple I*. Para ello os vamos a pedir que por vuestra cuenta busqueis un *dataset* que os guste, por ejemplo de Kaggle o de donde queráis extraerlo. De forma que vamos a probar a realizar ajustes lineales con el fin de predecir alguna de sus variables. Este *dataset* será el que iremos usando a lo largo de todas las sesiones de *pairprogramming* de regresión lineal.

Se ruega a la hora de realizar la entrega que incluyais el conjunto de datos que hayais decidido emplear para estos ejercicios.

Objetivos

Los objetivos de hoy son:

1. Buscar un conjunto de datos a analizar:

- Es necesario que el conjunto de datos a analizar tenga variables numéricas y categóricas, primando que haya más de una variable de tipo numérico. Ya que también se utilizarán serán útiles a la hora de realizar predicciones en las clases de regresión lineal múltiple. Así mismo se recomienda que los datasets estén ligeramente limpios para facilitar los análisis.

2. Explicar los datos y las variables disponibles en el conjunto de datos seleccionado:

- Esto se hace para que comprendáis mejor los datos con los que estáis trabajando.
- Realizar un EDA sencillo para la variable dependiente y la variable independiente.

Happy coding

Test Estadísticos

En esta primera lección de test estadísticos haremos una aproximación a este concepto. Cuáles son los principales test que tenemos y cuando los podremos usar.

Test estadísticos I - Introducción

Estos test son técnicas explicativas que utilizan muestras representativas de una población (de nuestros datos) para comprobar la certeza de nuestras afirmaciones (llamadas hipótesis). Esta certeza se expresa en términos de probabilidad.

Aprenderemos que es:

- La hipótesis nula
- Error de Tipo I y Tipo II
- p-valor



Regresion-Lineal-II-Test-Estadisticos.ipynb 101KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso

- **error de tipo I** rechazamos H_0 siendo verdadera
- **error de tipo II** aceptamos H_0 siendo falsa

Ejemplo 1

Supongamos que H_0 es: El equipo de escalada de Silvia es seguro Supongamos que H_1 es: El equipo no es seguro

- Error de Tipo I: silvia piensa que su equipo puede no ser seguro cuando en realidad si lo es.
- Error de Tipo II: pensamos que el equipo es seguro, pero en realidad no lo es.

Es mejor equivocarse en el error de tipo I, por que sino Silvia muere

Ejemplo 2

Supongamos que H_0 es: El análisis de sangre no contiene rastros de un patógeno

Supongamos que H_1 es: El análisis de sangre si contiene rastros de un patógeno

- Error de Tipo I: Lola está enferma pero en realidad no lo está
- Error de Tipo II: Lola no está enferma pero en realidad si lo está

En mejor equivocarse en el error de tipo I.

Pero en que error es mejor equivocarse depende de como este construida nuestra hipótesis nula.

Resumido en una tabla

H0	Verdadera	Falsa	Text
Aceptar	Decisión correcta	Error de Tipo II	
Rechazar	Error de Tipo I	Decisión correcta	

test shapiro

- H0 datos normales
- H1 datos no son normales

p-valor nivel significación, probabilidad de rechazar H0 siendo cierta.

- p-valor < 0.05 rechazar la H0 --> los datos no son normales.
- p-valor > 0.05 aceptamos la H0 --> los datos son normales.

Pair Programming Test Estadísticos

En este ejercicio de *pair programming* seguiréis usando el *dataset* que seleccionasteis ayer.

1. ¿Qué diferencia existe el error de tipo I y el error de tipo II?
2. ¿Qué es la asimetría?
3. ¿Es vuestra variable respuesta asimétrica?
4. ¿Qué tipo de curtosis presenta vuestra variable respuesta?
5. ¿Es vuestra variable respuesta normal? Demostrarlo visual y analíticamente.

Happy coding

Covarianza y Correlación

En esta lección aprenderemos que es la correlación y la covarianza. Como podemos hacerlo sobre un dataset y como interpretar los resultados.

A menudo nos interesa observar y medir la relación entre 2 o más variables. Es en este momento cuando podemos encontrar conceptos como la `covarianza` y la `correlación`.

- La `covarianza` indica si ambas variables varían en la misma dirección (**covarianza positiva**, es decir, si aumenta el valor de una variable aumenta el valor de la otra) o en dirección opuesta (**covarianza negativa** cuando aumenta el valor de una de las variables, disminuye el de la otra). En este caso valor de la covarianza no es importante, solo el signo (si es positivo o negativo)
- La `correlación` varía entre -1 y 1. Dos variables están asociadas o correlacionadas cuando una variable nos da información acerca de la otra. De la misma que la covarianza nos va a indicar la si existe relación entre dos variables. Al contrario que la covarianza, la correlación no nos indica únicamente la dirección (positiva o negativa), también nos indica la "fuerza" de esa relación. De esta forma:



Regresion-Lineal-III-Correlación&Covarianza.ipynb 2MB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Correlación y covarianza

En este ejercicio de *pair programming* seguiréis usando el *dataset* que seleccionasteis ayer.

1. Pregunta teórica: ¿Qué diferencia existe entre la covarianza y la correlación?
2. Calculad la covarianza y la correlación de nuestro *dataset*. No nos vale solo con que la calculéis, debéis hacer una interpretación de los resultados obtenidos. Calculad solo la correlación de Pearson.
3. Con los resultados de la correlación del ejercicio anterior, cread un `heatmap`.
4. Guardad la gráfica del `heatmap`.

Happy coding

Asunciones Regresión Lineal

En esta lección aprenderemos que es un ANOVA y que asunciones deben cumplirse para hacer un análisis de este tipo.

Asunciones Regresión Lineal

Las asunciones cuando queramos trabajar con un ANOVA:

Sin embargo, no nos podemos poner a hacer nuestras operaciones a lo loco. Antes de hacer nada, tenemos que comprobar una serie de asunciones. ¿Cuáles son?

- Los datos deben ser normales -> test de normalidad
- Las variables deben ser independientes -> correlación
- Homogeneidad de las varianzas o heterocedasticidad -> test de heterocedasticidad



Regresion-Lineal-IV-Asunciones.ipynb 431KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Repaso

Asunciones

- Normalidad
- Independencia de variables
- Homocedasticidad

Normalidad

- La vamos a testar sobre la variable respuesta
 - Análisis:
 - Shapiro test: cuando la muestra sea menor de 5000. O tamaños muestrales pequeños
 - El ruso: cuando la muestra grande.
 - Resultados:
 - H_0 : datos normales
 - H_1 : datos no normales
- p-valor < 0.05 ==> Datos no normales
 p-valor > 0.05 ==> Datos normales
- Que pasa cuando los datos no son normales:
 - Transformaciones de la VR --> aprenderemos en la lección de normalización.
 - Tendremos que usar otro tipo de algoritmo: Decision Tree, Random Forest

Independencia de las variables

- Lo vamos a testar sobre las predictoras entre si.
 - Análisis:
 - Si son variables numéricas --> correlación de Pearson
 - Si son variables categóricas --> Chi-cuadrado con la V-Cramer
 - Resultados var. categóricas:
 - H_0 : las variables son independientes
 - H_1 : las variables son dependientes
- p-valor < 0.05 ==> rechazamos H_0 == Variables dependientes.
 p-valor > 0.05 ==> no podemos rechazar H_0 == Variables son independientes.
- Resultados var. numéricas:
 - valores cercanos a 0 ==> no hay correlación
 - valores cercanos a 1 ==> correlación positiva
 - valores cercanos a -1 ==> correlación negativa
- p-valor < 0.05 ==> rechazamos H_0 == Variables dependientes.
 p-valor > 0.05 ==> no podemos rechazar H_0 == Variables son independientes.
- ¿Qué haremos si tenemos variables dependientes?
 - Filtrado de las variables y quedarnos con aquellas que son más importantes para nuestro modelo. En base a mi conocimiento previo del problema.

Homogeneidad de varianza

- Lo vamos a testar sobre las predictoras.
 - Recordamos que era la varianza: la desviación típica al cuadrado. Cuanto se desvía de media los datos de la media de la columna (la variable)
 - Análisis:
 - Visualmente:
 - Si las variables son categóricas: boxplot o violin plot
 - Si las variables son numéricas: regplot
 - Analiticamente:
 - Test de Levene
 - Test de Barlet
 - Resultados:
 - H0: las varianzas son iguales, son homogeneas, homocedasticidad
 - H1: las varianzas son distintas, son heterogeneas, heterocedasticidad
- p < 0.05 --> rechazamos H0 ==> heterocedasticidad p > 0.05 --> aceptamos H0 ==> homocedasticidad

Links interesantes

- [video explicación normalidad](#)
- [link bases asunciones -->](#)

Pair Programming Asunciones Regresión Lineal

En este ejercicio de *pair programming* seguiréis usando el *dataset* que seleccionasteis.

El objetivo de este *pair programming* es que evaluéis si vuestro set de datos cumple todas las asunciones que se deben cumplir para hacer una regresión lineal. Recordamos que estas asunciones son:

- Normalidad (ya la deberíais haber evaluado)
- Homogeneidad de varianzas
- Independencia de las variables

Cada asunción la deberéis testar de forma visual y analítica.

Happy coding

Normalización

En este jupyter aprenderemos qué es la normalización. Cómo podemos hacerlo desde Python y entendiendo la importancia de aplicar estas transformaciones en nuestros datos.

El objetivo de la normalización es cambiar los valores de las columnas numéricas en el conjunto de datos a una escala común, sin distorsionar las diferencias en los rangos de valores. Solo se requiere cuando las funciones tienen diferentes rangos. Pongamos un ejemplo, imaginemos que tenemos un *DataFrame* que tiene dos variables numéricas (dos columnas): precio y litros. El precio varía entre 1-2, mientras que los litros varían entre 0-90. Los litros son unas 50 veces mayor que el precio. Estas dos variables están en rangos muy diferentes. Cuando hacemos más análisis, los litros influirán intrínsecamente más en el resultado debido a su mayor valor. Pero esto no significa necesariamente que sea más importante como variable explicativa. Entonces normalizamos los datos para traer todas las variables al mismo rango.



Regresion-Lineal-V.1-Normalizacion.ipynb 399KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Normalización

En este ejercicio de *pair programming* seguiréis usando el *dataset* que seleccionasteis.

En vuestro *dataset* habréis identificado unas variables predictoras y una variable respuesta. Los objetivos del *pair programming* de hoy son:

- Sobre la variable respuesta, en *pair programmings* anterior identificastéis si seguía una distribución normal o no. En caso de que no siguiera una distribución normal, normalizarla. Podéis usar el método que prefiráis o el que mejor se ajuste
- Guardar en un csv el nuevo *dataframe* que habéis creado para seguir usándolo en los siguientes *pair programmings*

Happy coding

Estandarización

En este jupyter aprenderemos qué es la estandarización. Cómo podemos hacerlo desde Python y entendiendo la importancia de aplicar estas transformaciones en nuestros datos.

En la estandarización ajustamos los valores de las variables con una media de 0 y con una desviación estándar de 1. De esta forma ajustaremos los valores al mismo rango (de una forma similar a la normalización).

Por ejemplo, una variable que oscila entre 0 y 1000 tendrá más peso que una variable que oscila entre 0 y 1. El uso de estas variables sin estandarización dará a la variable con el rango más grande un peso de 1000 en el análisis. Transformar los datos a escalas comparables puede evitar este problema. Los procedimientos típicos de estandarización de datos igualan el rango y / o la variabilidad de los datos.



Regresion-Lineal-VI-Estandarizacion.ipynb 418KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Normalización

En este ejercicio de *pair programming* seguiréis usando el *dataset* que seleccionasteis.

En vuestro *dataset* habréis identificado unas variables predictoras y una variable respuesta. Los objetivos del *pair programming* de hoy son:

- Sobre las variables predictoras, en este caso deberéis estandarizar estas variables. De nuevo, usad el método que prefiráis.
- Guardar en un csv el nuevo *dataframe* que habéis creado para seguir usándolo en los siguientes *pair programmings*

Happy coding

ANOVA

En esta lección aprenderemos como hacer un ANOVA y un MANOVA usando Python y como interpretar los resultados.

ANOVA

El análisis de la varianza (ANOVA) proporciona una prueba estadística de si las medias de varios (k) grupos son iguales, y por tanto generaliza la prueba t a más de dos grupos. Los ANOVA's son útiles para comparar tres o más medias (grupos o variables) en busca de significación estadística.

Para poder hacer un ANOVA tenemos que tener en cuenta que se deben de cumplir una serie de asunciones o supuestos aprendidas en la lección anterior.

MANOVA

MANOVA es una prueba que analiza la relación entre varias variables de respuesta y un conjunto común de predictores al mismo tiempo. Por lo tanto, es igual al ANOVA, excepto que utiliza dos o mas variables respuesta.

MANOVA tiene algunas ventajas importantes en comparación con la ejecución de múltiples análisis de ANOVA, una variable de respuesta a la vez, que veremos durante la lección.



Regresion-Lineal-VII-ANOVA.ipynb 42KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming ANOVA

En el *pair programming* de hoy usaremos el set de datos que guardastéis en el *pair programming* de normalización y estandarización.

Hasta ahora habéis estado evaluando las características de vuestro set de datos y habéis hecho una gran exploración, es el momento de hacer vuestro primer ANOVA! En el ejercicio de hoy tendréis que hacer un ANOVA con vuestro datos y hacer una interpretación de los resultados.

NOTA Puede que vuestros datos no se ajusten o no cumplan todas las asunciones, no pasa nada, haced el ANOVA e interpretad los resultados. En próximas lecciones aprenderemos que podemos hacer cuando nos encontramos en esta situación.

Happy coding

Encoding

En este jupyter veremos como procesar las variables categóricas.

En algunas ocasiones, nuestro *dataset* contendrá variables categóricas. Estas variables suelen almacenarse como valores de texto (*strings*). Algunos ejemplos son el color ("Rojo", "Amarillo", "Azul"), el tamaño ("Pequeño", "Mediano", "Grande") o las denominaciones geográficas (Estado o País).

Algunas de las características de este tipo de variables son:

- No se pueden medir numéricamente
- No otorga datos específicos y a veces tampoco un orden
- Especifica una condición, calidad o característica

Podemos encontrar tres tipos de variables cualitativas:

- **Ordinalia** : la variable cualitativa ordinaria, también conocida como variable cuasicuantitativa, es representada por una modalidad que no requiere números pero sí consta de un orden o un puesto.
Por ejemplo, el nivel socioeconómico: alto, medio, bajo.
- **Nominal** : variable que no es representada por números ni tiene algún tipo de orden, y por lo tanto es matemáticamente menos precisa.
Por ejemplo, son variables nominales los colores: negro, azul, rojo, amarillo, naranja, etc.
- **Binaria** : la variable cualitativa binaria trabaja con valores específicos del tipo binario.
Por ejemplo, el sexo de una persona será masculino o femenino.

Muchos algoritmos de *machine learning* pueden admitir valores categóricos sin más manipulación, pero hay muchos más algoritmos que no lo hacen. Por lo tanto, el analista se enfrenta al reto de averiguar cómo convertir estos atributos de texto en valores numéricos para su posterior procesamiento.

Como ocurre con muchos otros aspectos del mundo del análisis de datos, no hay una respuesta única sobre cómo abordar este problema. Cada enfoque tiene ventajas y desventajas y tiene un impacto potencial en el resultado del análisis. Afortunadamente, las herramientas Pandas y scikit-learn de Python proporcionan varios enfoques que se pueden aplicar para transformar los datos categóricos en valores numéricos adecuados.

En esta lección aprenderemos algunos de los diversos enfoques comunes para poder hacer una buena codificación o *encoding* de estas variables.



Regresion-Lineal-VIII-Encoding.ipynb 145KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Encoding

En el *pair programming* de hoy usaremos el set de datos que guardastéis en el *pair programming* de normalización y estandarización.

Vuestro set de datos debería tener al menos una variable categórica, el objetivo del *pair programming* de hoy:

- Hacer una codificación de la/las variables categóricas que tengáis en vuestro set de datos.
- Recordad que lo primero que deberéis hacer es decidir si vuestras variables tienen o no orden, para que en función de esto uséis una aproximación u otra.
- Guardad el dataframe, donde deberíais tener las variables estadandarizadas, normalizadas y codificadas en un csv para usarlo en el próximo *pairprogramming*

Happy coding

Regresión Lineal Intro

¿Qué es la regresión lineal? En esta lección presentamos los conceptos más básicos de lo que es los problemas de regresión y como realizar una regresión lineal simple.

Este contenido ya pertenece a lo que sería conocida comúnmente como la rama de ML o Machine Learning (aprendizaje máquina), en la cual se busca que un algoritmo aprenda por cuenta propia a realizar diferentes tareas, en este caso buscamos predecir los posibles valores de una variable, teniendo en cuenta unos valores con lo que se entrenará el algoritmo. Este es uno de los problemas principales dentro del campo del aprendizaje máquina.

Como *Data Analyst*, es posible que eventualmente tengais que realizar algun tipo de regresión lineal, aunque no será vuestro cometido principal. Ya que de estas tareas se suele encargar las llamadas *Data Scientist*



Regresion-Lineal-IX-Intro-Regresion-Lineal.ipynb 198KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Intro Regresión Lineal

En el *pair programming* de hoy debéis usar el csv que guardastéis cuando hicistéis el *pairprgramming* de codificación (este csv debería tener las variables estadandarizadas).

El objetivo de este *pairprogramming* es que hagáis vuestro primer modelo de *machine learning*. Para eso recordad que:

- Deberéis separar la X y la y.
- Deberéis crear el *train* y el *test*.
- Ajustar vuestro modelo.

Happy coding

Regresión Lineal Métricas

Hoy aprenderemos como evaluar lo bueno o malo que es un modelo de regresión lineal a través del uso de las métricas.

Hasta ahora habíamos aprendido que "normas" se deben de cumplir cuando queremos hacer un modelo de *machine learning* usando una regresión lineal. Pero de nada sirve hacer estas predicciones si no sabemos si nuestro modelo es bueno haciéndolas. En esta lección aprenderemos cuáles son las principales métricas y como podemos calcularlas en Python. Entre las más importantes encontramos:

En este jupyter aprenderemos cuales son las principales métricas de los modelos de regresión lineal. Estas métricas nos ayudarán a entender si nuestro modelo es el mejor para predecir.

Tras haber realizado el ajuste lineal podemos emplear diversas métricas o estimadores de la bondad (o lo que es lo mismo, como de bueno es nuestro modelo prediciendo nuevos datos) del ajuste realizado, las más utilizadas son las siguientes:

- **R²** : Es una medida estadística que representa la proporción de la varianza que puede ser explicada por las variables independientes (o predictoras) del modelo de regresión. Este indicador solo puede ser utilizado en regresiones lineales. Es decir, imaginemos que tenemos un R² de 67%. Eso significa que todas las variables predictoras explican un 67% de la variación que encontramos en la variable respuesta, lo que es lo mismo que un 33% de la variación no está explicado por ninguna de las variables que tenemos, lo que sugiere que otras variables que nosotras no hemos considerado podrían explicar esa parte.
- **MAE (Mean absolute error)**: sirve para obtener una medida de la diferencia entre los valores predichos frente a los reales. Nos indicará la precisión de la predicción obtenida. A menor MAE, mejor es capaz de ajustar los datos del modelo que hemos creado.
- **MSE (Mean Squared Error)**: mide el promedio(media) de los errores al cuadrado.A menor MAE, mejor es capaz de ajustar los datos del modelo que hemos creado.
- **RMSE (Root Mean Squared Error)**: nos muestra la distancia promedio entre los valores predichos y los valores reales del dataset. A menor RMSE, mejor es capaz de ajustarse el modelo obtenido.



Regresion-Lineal-X-Metricas-Regresion-Lineal.ipynb 184KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Métricas Regresión Lineal

En el *pair programming* anterior creastéis vuestro primer modelo de *machine learning* usando la regresion Lineal. Es el momento, que con vuestros datos evaluéis si es bueno haciendo predicciones. Los objetivo de este *pairprogramming* son:

- Calculéis las métricas para vuestro modelo
- Discutid los resultados de las métricas y extraed conclusiones
- Guardad los resultados de las métricas en un csv para usarlo más adelante.

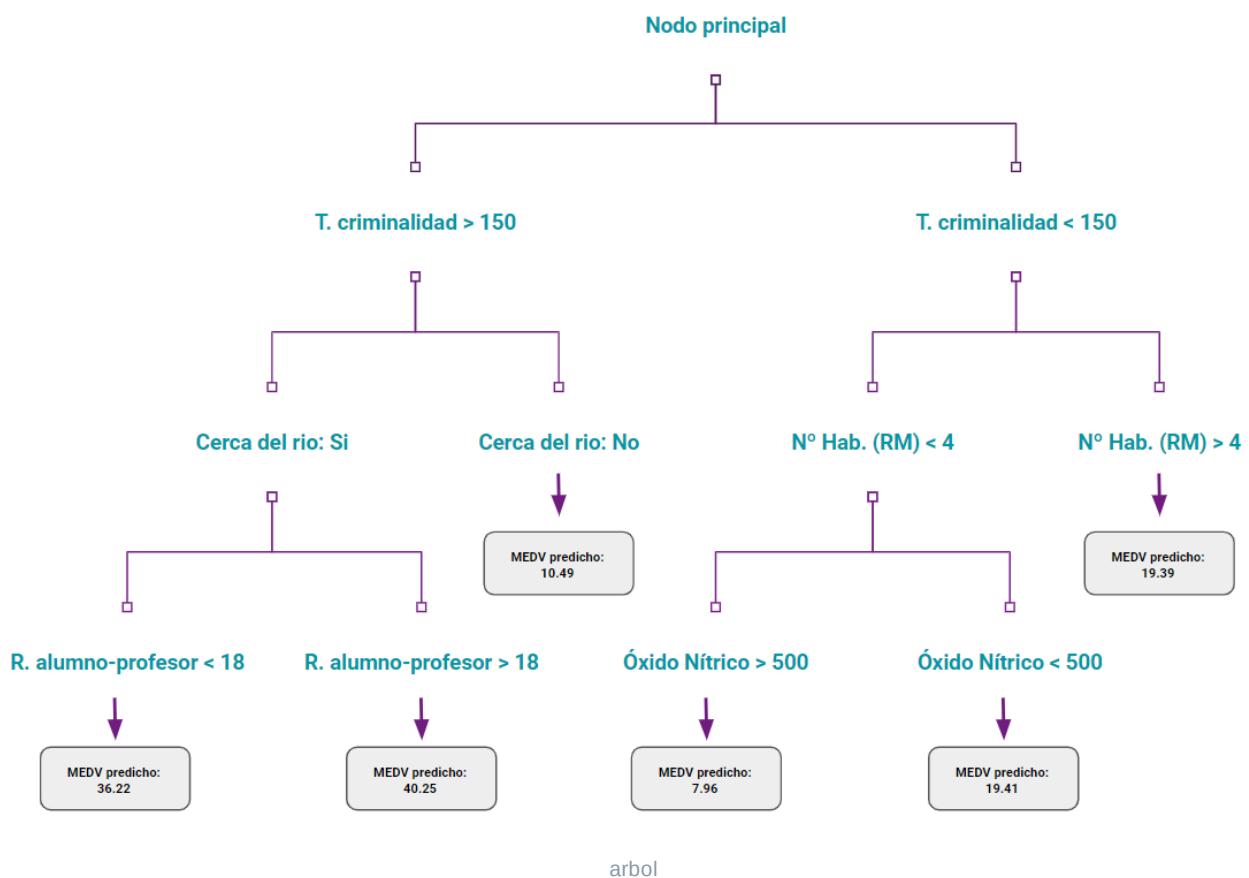
Happy coding

Decision Tree

¿Qué es Decision Tree? En esta lección presentamos los conceptos más básicos de un Decision Tree y como podemos hacerlo en Python.

Crear un árbol de decisiones es en un proceso de dividir los datos de entrada, este es un procedimiento numérico en el que se alinean todos los valores y se prueban diferentes puntos de división utilizando distintos métodos. Todas las variables de entrada y todos los puntos de división posibles se evalúan y se elige la que tenga mejor resultado.

A forma de esquema un árbol de decisión se podría ver visualmente así:



Regresion-Lineal-XI-Decision-Tree.ipynb 546KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Decision Tree

En el *pair programming* de hoy debéis usar el csv que guardastéis cuando hicistéis el *pairprgramming* de codificación (este csv debería tener las variables estadandarizadas).

En *pairprogramming* anteriores ajustastéis vuestro datos a una regresión lineal. El objetivo de hoy es:

- Ajustar el modelo a un *Decision Tree*
- Extraer las métricas
- Debatid entre vosotras que modelo es mejor y por qué (basándose en las métricas)

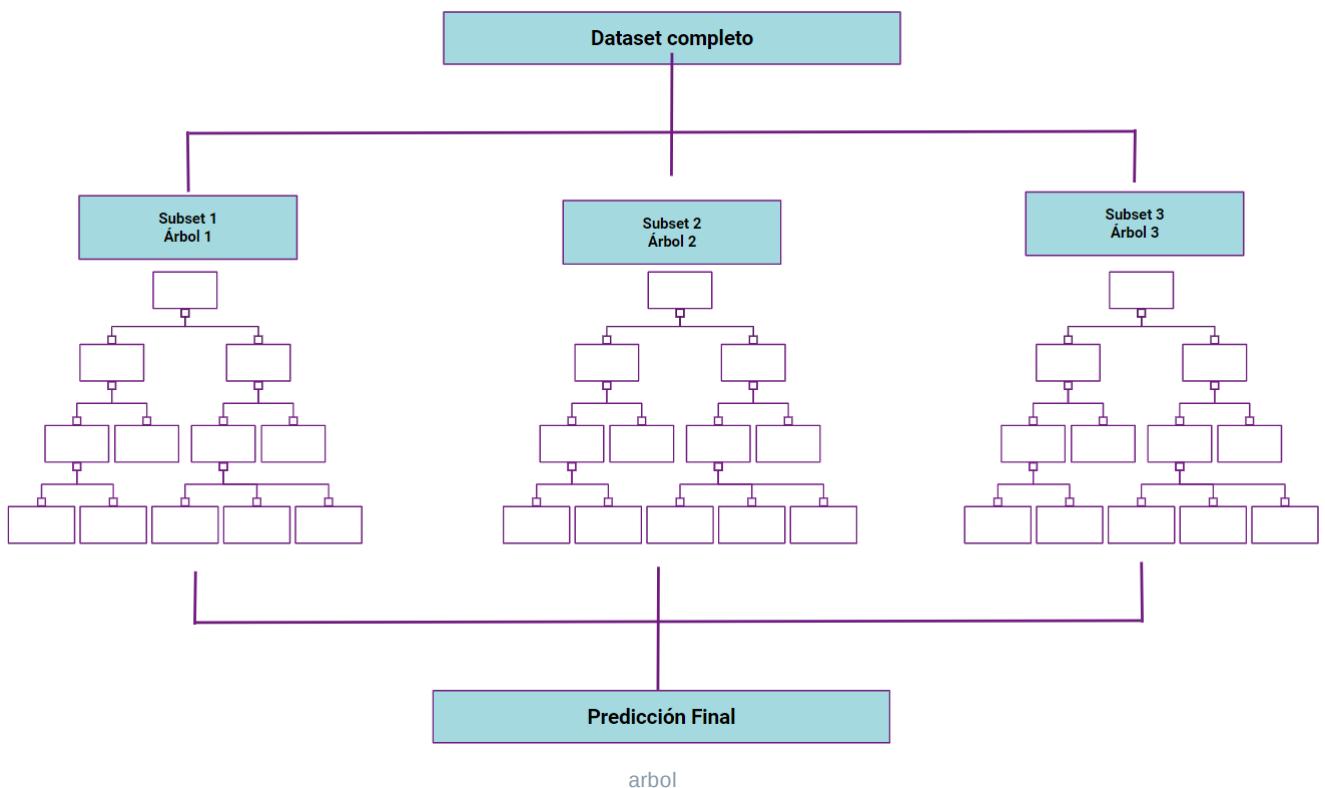
Happy coding

Random Forest

¿Qué es Random Forest? En esta lección presentamos los conceptos más básicos de un Random Forest y como podemos hacerlo en Python.

Un modelo *Random Forest* está formado por un conjunto de árboles de decisión individuales, cada uno entrenado con una muestra ligeramente distinta de los datos de entrenamiento generada mediante (*bootstrapping*). La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo

A forma de esquema un se podría ver visualmente así:



Regresion-Lineal-XII-Random-Forest.ipynb 40MB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Decision Tree

En el *pair programming* de hoy debéis usar el csv que guardastéis cuando hicistéis el *pairprgramming* de codificación (este csv debería tener las variables estadandarizadas).

En *pairprogramming* anteriores ajustastéis vuestro datos a una regresión lineal y a un *Decision Tree*. El objetivo de hoy es:

- Ajustar el modelo a un *Random Forest*
- Extraer las métricas
- Debatid entre vosotras que modelo es mejor y por qué (basándose en las métricas)

Happy coding

Repaso Regresión Lineal

Hoy haremos un repaso de los principales conceptos de Regresión Lineal aprendidos hasta el momento

Hoy seremos científicas de datos que trabajamos para una compañía de seguros, nuestro jefe tiene unos datos sobre el dinero que gastan en una serie de clientes basándose en una serie de características, estas son:

- edad: edad del beneficiario principal
- sexo: género del contratante del seguro, mujer, hombre
- bmi: índice de masa corporal, que proporciona una comprensión del cuerpo, los pesos que son relativamente altos o bajos en relación con la altura, índice objetivo de peso corporal (kg / m^2) utilizando la relación entre la altura y el peso, idealmente 18,5 a 24,9
- niños: Número de hijos cubiertos por el seguro médico / Número de personas a cargo
- fumador: Fumador
- región: la zona de residencia del beneficiario en los EE.UU., noreste, sureste, suroeste, noroeste.
- gastos: Gastos médicos individuales facturados por el seguro de enfermedad

Nuestro objetivo, intentar predecir futuros gastos para nuevos clientes.



insurance.csv 59KB
Binary

Descarga este Jupyter y ábrelo en VS Code.



repaso-I-Regresion-Lineal.ipynb 814KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

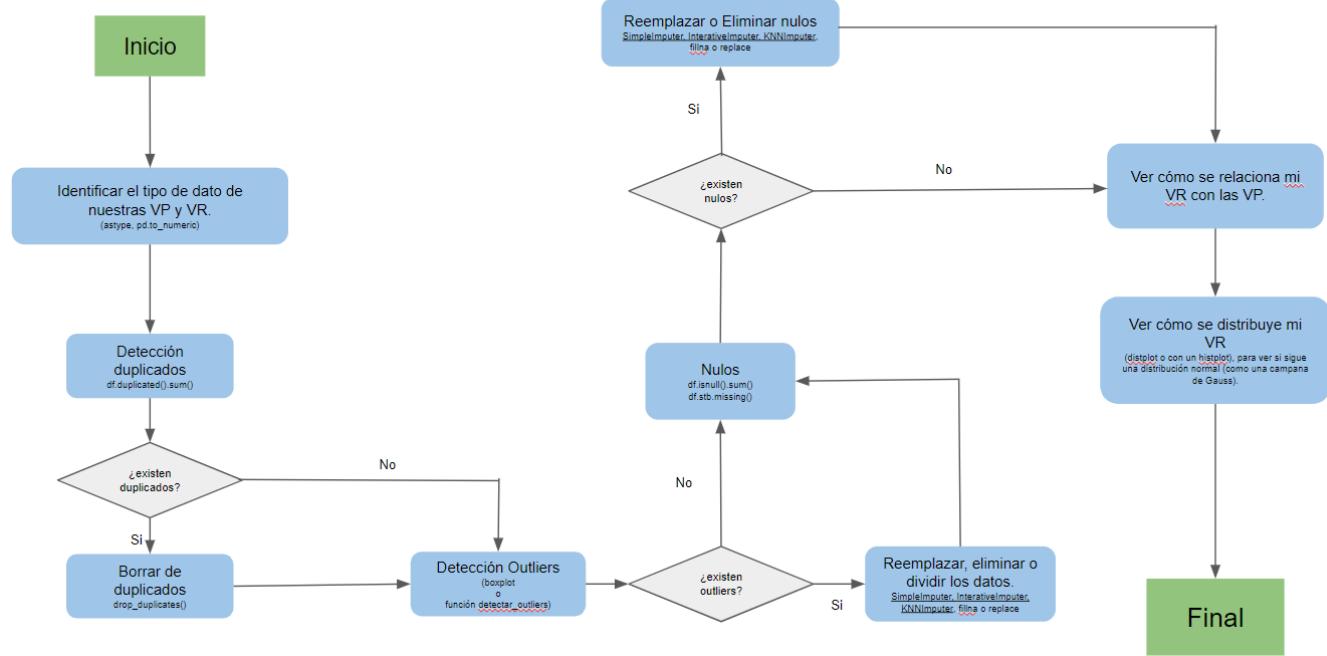
Links que podrían ser de interés

- [Entendiendo el test de la F](#)
- [Entendiendo el test de la t](#)
- [Entendiendo el anova](#)
- [Entendiendo la regresión lineal](#)

Flujo de trabajo EDA y ANOVA

EDA

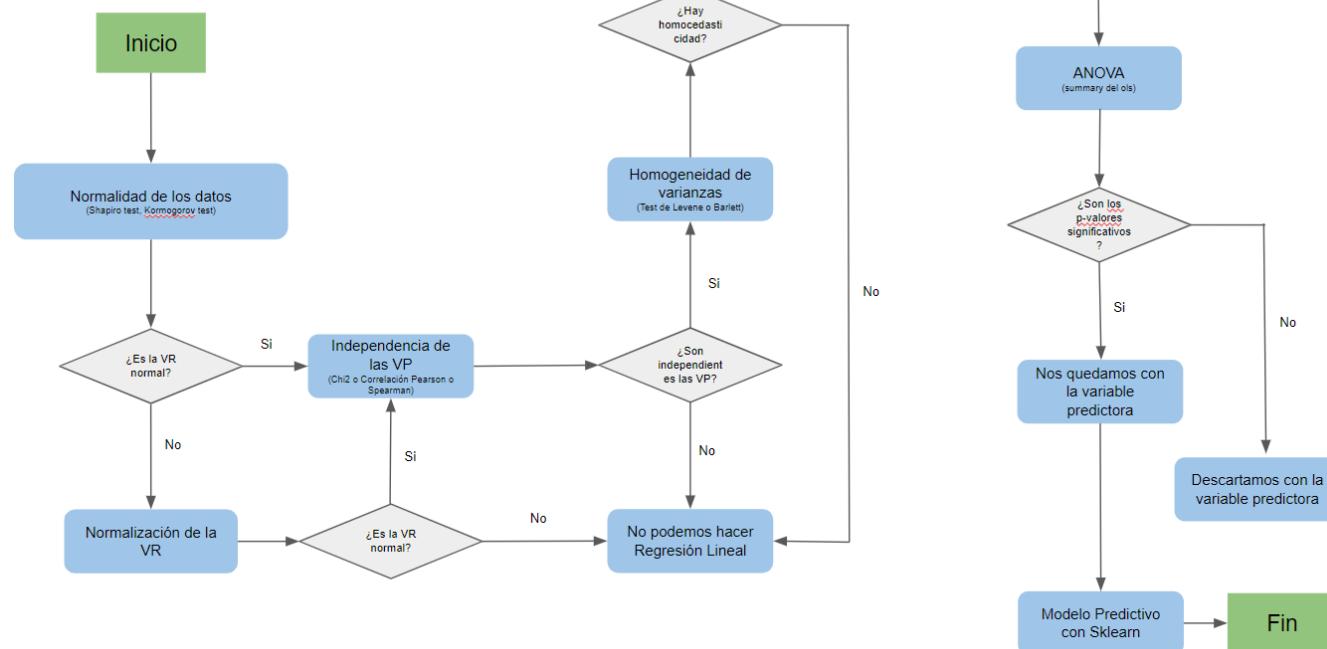
Paso 1: EDA



eda

Asunciones ANOVA

Paso 2: Asunciones



anova

Regresión Logística

EDA

En la lección nos enfremos a un nuevo problema de machine learning, los problemas de regresión logística. Haremos una introducción a que es la regresión logística y haremos un EDA para el set de datos

Como ya sabemos, antes de ponernos manos a la obra, como analistas de datos necesitamos saber como son nuestros datos, como se relacionan las variables, si tenemos nulos, gestionar los *outliers*, etc.

En esta lección trabajaremos con el csv del titanic. Nuestro objetivo, predecir si una persona va a sobrevivir o no en un accidente similar al del titanic.



titanic.csv 59KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



Regresion-Logistica-I-EDA.ipynb 458KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming EDA

Para los ejercicios de *pair programming* de Regresión logística temdremos que buscar un *dataset* (al igual que hicimos en regresión lineal) que usaremosmos a lo largo de los siguientes ejercicios.

Se ruega a la hora de realizar la entrega que incluyais el conjunto de datos que hayais decidido emplear para estos ejercicios.

Objetivos

- Buscar un conjunto de datos a analizar
Se recomienda que el conjunto de datos a analizar tenga variables numéricas y categóricas, primando que haya más de una variable de tipo numérico.
- Explicar los datos y las variables disponibles en el conjunto de datos seleccionado
- Realizar un EDA sencillo poniendo en práctica los conocimientos adquiridos hasta el momento.
- Interpretación de los resultados.

Happy coding

Preprocesado

En esta lección realizaremos los cambios oportunos para poder ejecutar el modelo de regresión logística.

Cuando nos enfrentamos a problemas de regresión lineal vimos que eran necesarios algunos cambios antes de poder ajustar los modelos. En el caso de la regresión logística lo tendremos que hacer. Estos cambios incluyen:

- Estandarización de las variables predictoras numéricas
- Codificación de las variables categóricas
- Balanceo de la variable respuesta



Regresion-Logistica-II-Preparacion-Datos.ipynb 126KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Preprocesado

Usando el mismo *dataset* que usatéis ayer, los objetivos de los ejercicios de hoy son:

- Estandarizar las variables numéricas de vuestro set de datos
- Codificar las variables categóricas. Recordad que tendréis que tener en cuenta si vuestras variables tienen orden o no.
- Chequear si vuestros datos están balanceados. En caso de que no lo estén utilizad algunas de las herramientas aprendidas en la lección para balancearlos.
- Guardad el *dataframe* con los cambios que habéis aplicado para utilizarlo en la siguiente lección.

Happy coding

Ajuste Regresión Logística

En esta lección aprenderemos como ajustar un modelo de regresión logística

Es el momento de ponernos manos a la obra. En esta lección ajustaremos nuestro primer modelo de regresión logística. *Don't panic!* Las cosas no cambian mucho, y usaremos la misma lógica que ya aprendimos con la regresión lineal como:

- Separar la X de la y
- Crear el set de datos de entrenamiento y de *test*
- Ajustar el modelo usando el método `.fit()`



Regresion-Logistica-III-Ajuste.ipynb 62KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Ajuste

Es el momento de realizar el ajuste de vuestro modelo, en este caso tendréis que usar el csv que guardastéis ayer después de todo el preprocesamiento. Los objetivos de esta lección son:

- Realizar el ajuste o ajustes de los modelos
- Sacad la matriz de confusión de vuestro modelo e identificad cuáles son los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

Happy coding

Métricas Regresión Logística

Hoy aprenderemos como evaluar lo bueno o malo que es un modelo de regresión logística a través del uso de las métricas.

Métricas de regresión logística

Al igual que en la regresión lineal, en la regresión logística tendremos una serie de métricas que nos darán información sobre como de bueno o malo es nuestro modelo haciendo predicciones. En este caso, las cosas cambian un poco, y las métricas son diferentes. Estas métricas son:

Métricas en la regresión logística.

Como ya anticipamos, las métricas que se han utilizado para las regresiones lineales no son aplicables para el caso de la regresión logística, por lo tanto deberemos introducir una serie de conceptos nuevos para ser capaces de evaluar los resultados obtenidos.

Vamos a ello!

- **Matriz de confusión:** donde nos encontraremos con los siguientes conceptos:
 - **Verdadero positivo (VP):** El valor se ha predicho como positivo y en la realidad es positivo.
 - **Falso Positivo (FP):** El valor ha sido predicho como positivo, cuando era negativo. Se ha predicho de forma incorrecta. (También llamado error de tipo I).
 - **Falso negativo (FN):** El valor ha sido predicho como negativo, cuando era positivo. Se ha predicho de forma incorrecta. (También llamado error de tipo II).
 - **Verdadero Negativo (VN):** El valor se ha predicho como negativo y en la realidad era negativo.

A raíz de la matriz de confusión podemos definir las siguientes métricas de la precisión de nuestras predicciones:

- **Accuracy:** también llamada **exactitud**. Mide que porcentaje de los valores predichos están bien predichos. Viene definida por la siguiente fórmula:
- **Recall:** también llamado **sensibilidad** o **exhaustividad**. Contesta a la siguiente pregunta: **¿Qué porcentaje de casos positivos fueron capturados?**. Viene definida por la siguiente fórmula
- **Precision:** contesta a la siguiente pregunta **¿Qué porcentaje de predicciones positivas fueron correctas?**
- **Especificidad :** es el opuesto a la exhaustividad o recall, es decir, contesta a la pregunta de **¿Qué porcentaje de casos negativos fueron capturados?**. Viene definida por la siguiente fórmula:
- **F1:** es la media de la *precision* y el *recall*, teniendo en cuenta ambas métricas en la siguiente ecuación:
- **kappa:** es una medida de concordancia que se basa en comparar la concordancia observada en un conjunto de datos, respecto a la que podría ocurrir por mero azar.



Regresion-Logistica-IV-Métricas.ipynb 71KB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Métricas

En el ejercicio de *pair programming* anterior ajustastéis vuestro primer modelito de regresión logística. Ahora es el momento de saber como de bueno es nuestro modelo. Para esto, los objetivos del *pair* de hoy son:

- Calculad las métricas para vuestro modelo
- Interpretad las métricas obtenidas, ¿es un buen modelo? ¿hay *overfitting* o *underfitting*?

Happy coding

Decision Tree

¿Qué es Decision Tree? En esta lección presentamos los conceptos más básicos de un Decision Tree y como podemos hacerlo en una regresión logística

De la misma forma que podemos aplicar este algoritmo a los problemas de regresión lineal, los podremos aplicar a los modelos de regresión logística.

Crear un árbol de decisiones es en un proceso de dividir los datos de entrada, este es un procedimiento numérico en el que se alinean todos los valores y se prueban diferentes puntos de división utilizando distintos métodos. Todas las variables de entrada y todos los puntos de división posibles se evalúan y se elige la que tenga mejor resultado.



Regresion-Logistica-V-Decision-Tree.ipynb 448KB
Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Decision Tree

Hasta ahora hemos ajustado el modelo usando una Regresión Logística, pero como hemos aprendido, podemos usar el *Decision Tree* en este tipo de problemas. Los objetivos de este *pair programming* :

- Ajustad un modelo de *Decision Tree* a nuestros datos.
- Calculad las métricas a nuestro nuevo modelo.
- Comparad las métricas con el modelo hecho hasta ahora. ¿Cuál es mejor?

Happy coding

Random Forest

¿Qué es Random Forest? En esta lección presentamos los conceptos más básicos de un Random Forest y como podemos hacerlo en Python.

Un modelo *Random Forest* está formado por un conjunto de árboles de decisión individuales, cada uno entrenado con una muestra ligeramente distinta de los datos de entrenamiento generada mediante (*bootstrapping*). La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo



Regresion-Logistica-VI-Random-Forest.ipynb 13MB

Binary

Descarga este Jupyter y ábrelo en VS Code.

Pair Programming Random Forest

Hasta ahora hemos ajustado el modelo usando una Regresión Logística, pero como hemos aprendido, podemos usar el *Random Forest* en este tipo de problemas. Los objetivos de este *pair programming* :

- Ajustad un modelo de *Random Forest* a nuestros datos.
- Calculad las métricas a nuestro nuevo modelo.
- Comparad las métricas con los modelos hechos hasta ahora. ¿Cuál es mejor?

Happy coding

Repaso Regresión Logística

En este apartado encontraréis el repaso de Machine Learning

Aquí encontraréis tanto el jupyter como los datos que usamos en el día de repaso.

- **Jupyter**



repaso_LogR_DT_RF.ipynb 1MB

Binary

[Descarga este Jupyter y ábrelo en VS Code.](#)



prediccion.ipynb 42KB

Binary

[Descarga este Jupyter y ábrelo en VS Code.](#)

- **Datos**



heart.csv 35KB

Binary

[Descarga este Jupyter y ábrelo en VS Code.](#)

Tableau

Datos

En este apartado os dejamos todo el archivos de datos que usaremos a lo largo de las sesiones de Tableau

- **Sesión teórica**

En estas sesiones necesitaremos los datos de `listings` y `neighborhoods`.



`listings.csv` 3MB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.



`neighbourhoods.geojson` 375KB

Binary

Descarga este fichero para poder hacer los ejercicios que os planteamos.

Tableau I - Introducción a Tableau

¿Qué es Tableau? ¿Cómo podemos insertar datos en Tableau? ¿Con qué tipos de ficheros podemos trabajar en Tableau?

Las dinámicas de estas lecciones cambiará un poco. En estas lecciones tendréis links a videos de Youtube y de la plataforma de aprendizaje Platzi donde os explicarán los conceptos que aprenderemos.

Pero antes de empezar deberéis:

1. Registraros en Tableau y creareis una cuenta. Es gratuita así que no habrá ningún problema.
2. Registraros en Platzi para poder acceder a algunos de sus videos que os iremos poniendo en las lecciones invertidas.

Tableau I - Intro a Tableau

Uno de los elementos más importantes a tener en cuenta a la hora de presentar un informe es el cuadro de mando (*Dashboard*). Para la creación de cuadros de mando existen una infinidad de herramientas de visualización de datos. Nosotras usaremos Tableau.

¿Qué es lo que hace de Tableau una herramienta tan usada?

Su funcionamiento es muy sencillo. Tableau permite arrastrar y soltar grandes cantidades de datos en un «lienzo» digital y realizar gráficas al instante. La idea detrás de cómo funciona Tableau es que es más fácil de manipular lo que está pasando en la interfaz para que se pueda ver lo que se está haciendo bien y, por extensión, lo que se está haciendo mal.

Cuenta con numerosas ventajas que le convierten en una herramienta de uso creciente. Podríamos destacar las siguientes:

- Su facilidad de uso, apto para usuarios que no tienen conocimiento alguno de programación.
- Su principal ventaja, además de la anterior, es por supuesto su planteamiento visual y sencillo de todo tipo de datos complejos. Así, se adapta a organizaciones de sectores muy diversos y con características muy diferentes.
- La sencilla conexión con fuentes de datos para realizar análisis.
- Es una herramienta perfecta para trabajar en equipo, ya que permite el sencillo acceso a los datos por parte de diversas personas.
- La posibilidad de integración en aplicaciones propias de los usuarios.

¿Por qué es importante hacer una buena visualización de los datos?

En este [link](#) podemos encontrar un pequeño video donde nos lo explican super bien. Pero si tuvieramos que explicarlo en una frase diríamos que una buena visualización:

Nos permite narrar de una forma sencilla una historia sobre nuestros datos. Ayuda a hacer comprender a nuestros interlocutores la historia que hay detrás de nuestros datos.

Productos:

Tableau tiene muchas herramientas, podemos destacar los 4 principales:

- **Tableau Desktop**: es la versión de escritorio y uno de los productos más destacados de Tableau. Permite realizar potentes visualizaciones desde nuestro equipo en pocos segundos. Es de pago, por lo que necesitaremos una licencia para poder trabajar con el.
- **Tableau Public**: es la versión gratuita de Tableau. Tendrá prácticamente todas las funcionalidades de Tableau Desktop. Podemos descargarlo en [este link](#). **Será el que usaremos a lo largo de estas sesiones**
- **Tableau Prep**: destinado principalmente al análisis de datos. Permite a los usuarios realizar el EDA (Exploratory Data Analysis) y construir flujos de datos de una forma rápida y segura
- **Tableau On-Line**: se trata de una plataforma de análisis alojada en la nube. Desde Tableau on-line se pueden compartir visualizaciones y cuadros de mando con la comunidad de Tableau. Es accesible tanto desde navegador web en un equipo de escritorio como en smartphone.
- **Tableau Server**: al igual que Tableau on-line permite compartir cuadros de mando. La principal diferencia es que nos permite guardar todo el desarrollo en nuestro servidor propio, como en los servidores en la nube de Tableau. Estos nos garantizan una rápida implementación, integración y simplicidad en cuanto a escalabilidad se refiere.

¿Cómo nos instalamos Tableau Public en nuestro ordenador?

En este [link](#) podréis ver como instalar la aplicación de Tableau Public para que podamos trabajar de aquí en adelante.

Conociendo la interfaz de Tableau

En [este](#) video podréis aprender como es la página principal de Tableau para empezar a familiarizarnos con la herramienta.

Medidas y Dimensiones

Al conectarse a una fuente de datos nueva, Tableau asigna cada campo de la fuente de datos como dimensión o medida del panel Datos, en función del tipo de datos que contenga el campo. Estos campos se utilizan para crear vistas de los datos.

Los campos de datos se crean a partir de las columnas de la fuente de datos. A cada campo se le asigna automáticamente un tipo de datos (por ejemplo, entero, cadena, fecha, etc.). Por otro lado se les asigna un "rol":

- **Dimensiones** contienen valores cualitativos (por ejemplo, nombres, fechas o datos geográficos). Puede utilizar las dimensiones para categorizar, segmentar y revelar los detalles de los datos. Las dimensiones afectan al nivel de detalle de la vista.
- **Medidas** contienen valores numéricos cuantitativos que se pueden medir. Las medidas se pueden agregar. Al arrastrar una medida a la vista, Tableau aplica una agregación a esa medida (de forma predeterminada).

Tableau representa los datos de manera diferente en la vista en función de si el campo es discreto (azul) o continuo (verde).

- Las medidas verdes y las dimensiones son continuas. Los valores de los campos continuos se tratan como un intervalo infinito. Por lo general, los campos continuos añaden ejes a la vista.
- Las medidas azules y las dimensiones son discretas. Los valores discretos se tratan como finitos. Por lo general, los campos discretos añaden encabezados a la vista.

¿Cómo cargamos datos en Tableau?

A lo largo de todas las lecciones de Tableau trabajaremos con dos ficheros que nos dan información sobre el alquiler de casas de Airbnb en Madrid. En concreto tendremos los siguientes datos:

- `listings.csv` : contiene la información de los alquileres de Airbnb en Madrid. Donde están, que puntuaciones tienen, quien nos lo alquila, etc.
- `neighbourhoods.geojson` : contiene la información geográfica de los barrios de Madrid.

Para entender como podemos cargar datos en Tableau debéis ver [este](#).

Además, tenéis [este](#) video donde os explicamos como cargar varias fuentes de datos en el mismo proyecto.

Nota En Tableau podremos meter como máximo 32 tablas diferentes!!

Al final de todas las lecciones, tendremos una vista con todos los datos de los alquileres de Airbnb en Madrid. Y podremos ver algo como esto

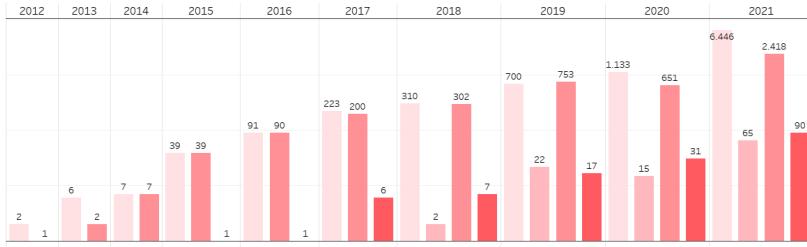


Room Type
 • Entire home/apt
 • Hotel room
 • Private room
 • Shared room

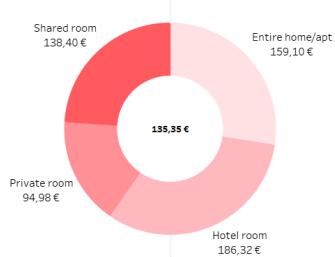
Evolución del precio



Oferta anual por tipo de alojamiento



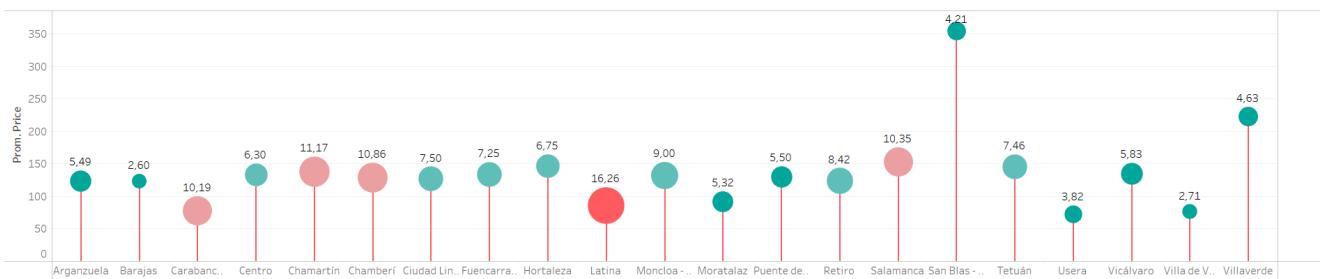
Precio medio por tipo de habitación



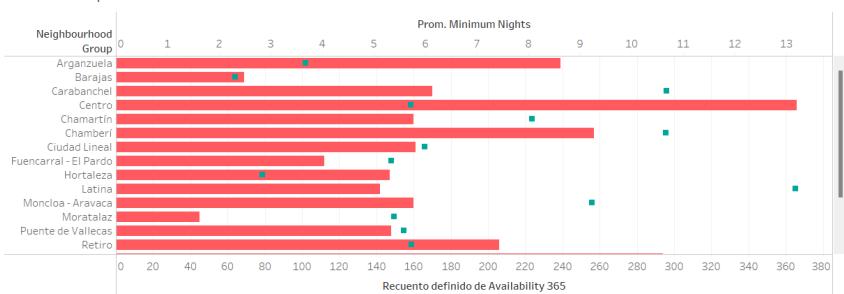
barrios-reservas hosts mínimo_noches precio-promedio-disponibilidad tipo-habitación oferta_anual_habitaciones mapa-precio-barato precio_medio_anual/habitaciones tipos-hab barrios Historia 1 mapas gráfico_lineas dash1



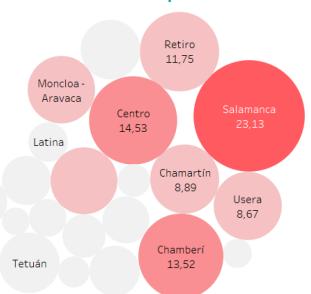
¿Qué barrio es más caro?



barrios-disponibilidad-noches



Media de reservas por barrio



barrios-reservas hosts mínimo_noches precio-promedio-disponibilidad tipo-habitación oferta_anual_habitaciones mapa-precio-barato precio_medio_anual/habitaciones tipos-hab barrios Historia 1 mapas gráfico_lineas dash

dash

Hasta aquí la primera lección de Tableau .

Tableau-I-Intro.md 7KB

Binary

Descarga este fichero para tener la lección.

Pair Programming Tableau

En estos ejercicios de *pair programming* "volaréis" un poco más solas. ¿Qué tendréis que hacer?

- Elegid un *dataset* que os guste, con ese mismo *dataset* estaréis trabajando durante todas las sesiones de *pair* de esta parte del *bootcamp*.
- Según vayan pasando las lecciones tendréis que ir aplicando los conocimientos aprendidos, creación de distintos tipos de gráficas, elegir paletas de colores adecuadas, crear campos calculados, montar vuestras *dashboards* y vuestras historias.

Nota El límite es el cielo, sed creativas, experimentad y disfrutad!!

Tableau II - Primeras Gráficas

¿Qué tipos de gráficas tenemos en Tableau? ¿Cómo podemos elegir que gráfica hacer? ¿Cómo podemos hacer mapas en Tableau?

¿Cómo podemos saber que tipo de gráfica hacer

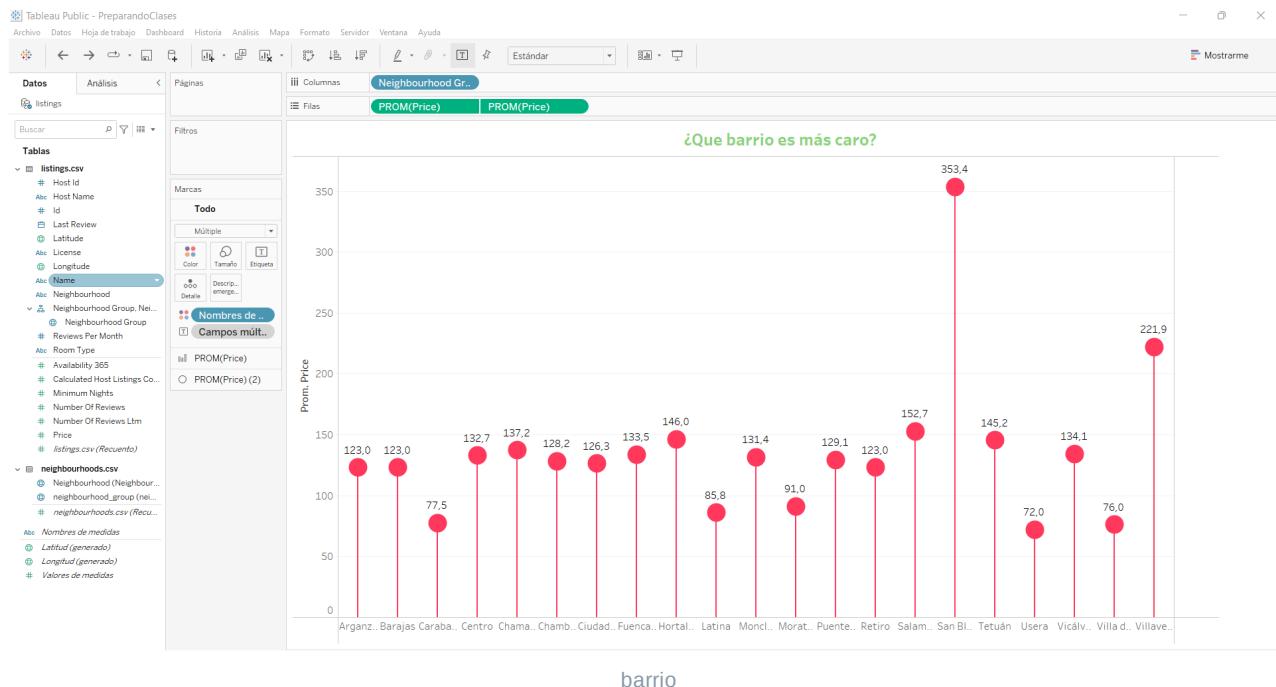
En [este](#) link podréis ver todos los gráficos que podemos usar en Tableau y cuando es mejor usar cada uno de ellos.

A modo de resumen podemos decir que los principales gráficos son:

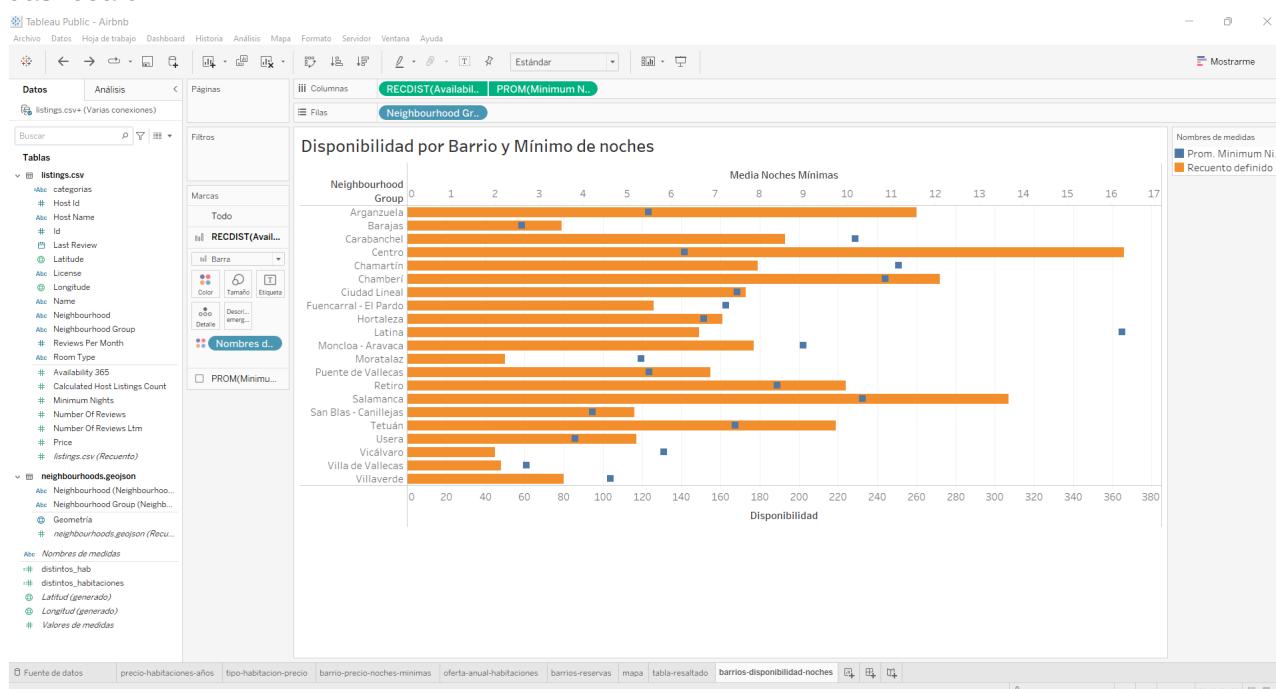
1. Gráfico de barras: los usaremos para mostrar datos de tipo categórico, organiza los datos en barras rectangulares que se pueden usar fácilmente para comparar conjuntos de datos. Puede crear un gráfico de barras si desea comparar dos o más valores de datos de un tipo similar y **si no tenemos demasiados grupos de datos para mostrar**, es decir, muchas categorías. Imaginamos que queremos ver el número de habitantes por comunidad autónoma en España, este podría ser un buen gráfico que usar.

Una solución para este caso es usar un filtro sobre nuestra gráfica, para que nos muestre solo aquellas ciudades que tengan más de un número dado de habitantes.

Después de [este video](#) deberíais obtener un gráfico similar a este . Los colores serán diferentes, pero no os preocupéis, un poco más adelante os enseñaremos como cambiarlos!



También podéis ver [este video](#) para tener todos los gráficos que necesitaremos para montar nuestro *dashboard*.



2. Gráfico mapa de árbol: se utiliza un mapa de árbol para demostrar diferentes partes de los datos en

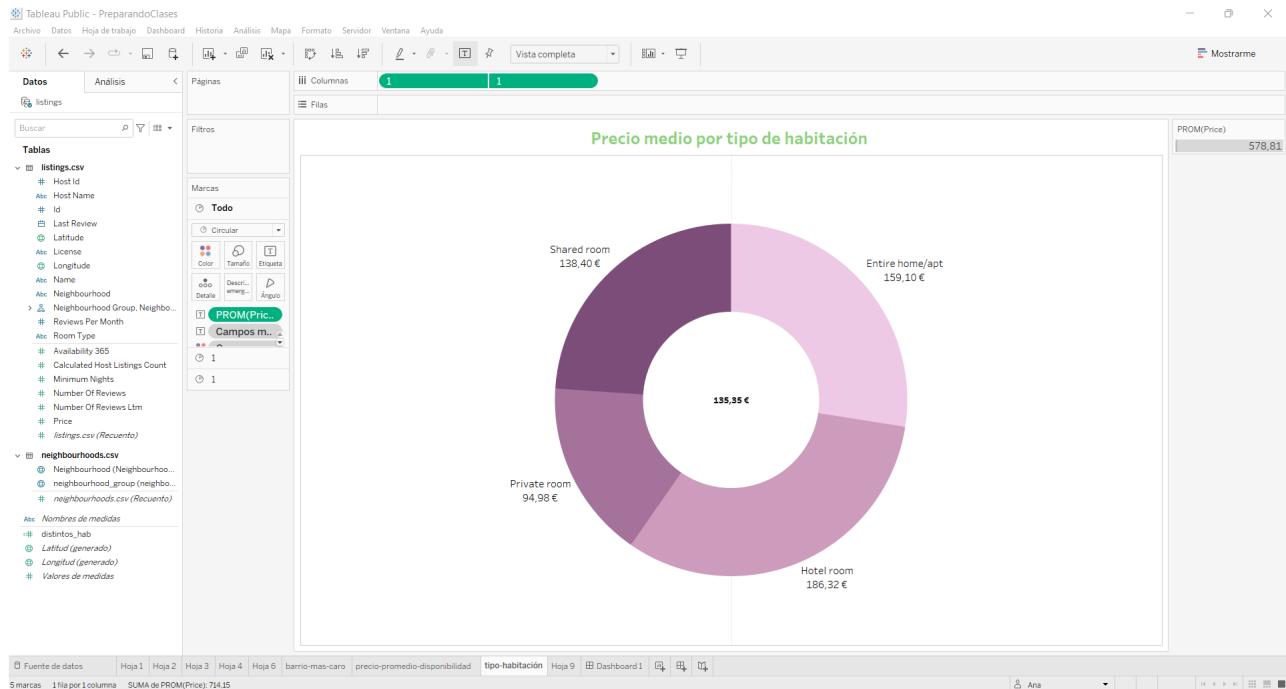
relación con el todo. Cada rectángulo en un mapa de árbol se divide en rectángulos más pequeños según sus proporciones a los datos completos, como las ramas de un árbol. Entonces, el mapa de árbol total muestra los datos completos, mientras que los rectángulos individuales muestran los datos secundarios en proporción al total.

3. **Gráfico circular:** se pueden utilizar para mostrar proporciones de nuestros datos. Si bien este tipo de gráfico no es el más usado, para ciertas cosas es muy útil, como por ejemplo mostrar el precio de cada tipo de habitación en nuestros datos. Para aprender a crearlos, ver los siguientes videos:

[Video-Piechart1](#)

[Video-Piechart2](#)

Después de ver los dos videos deberíais tener un gráfico como este

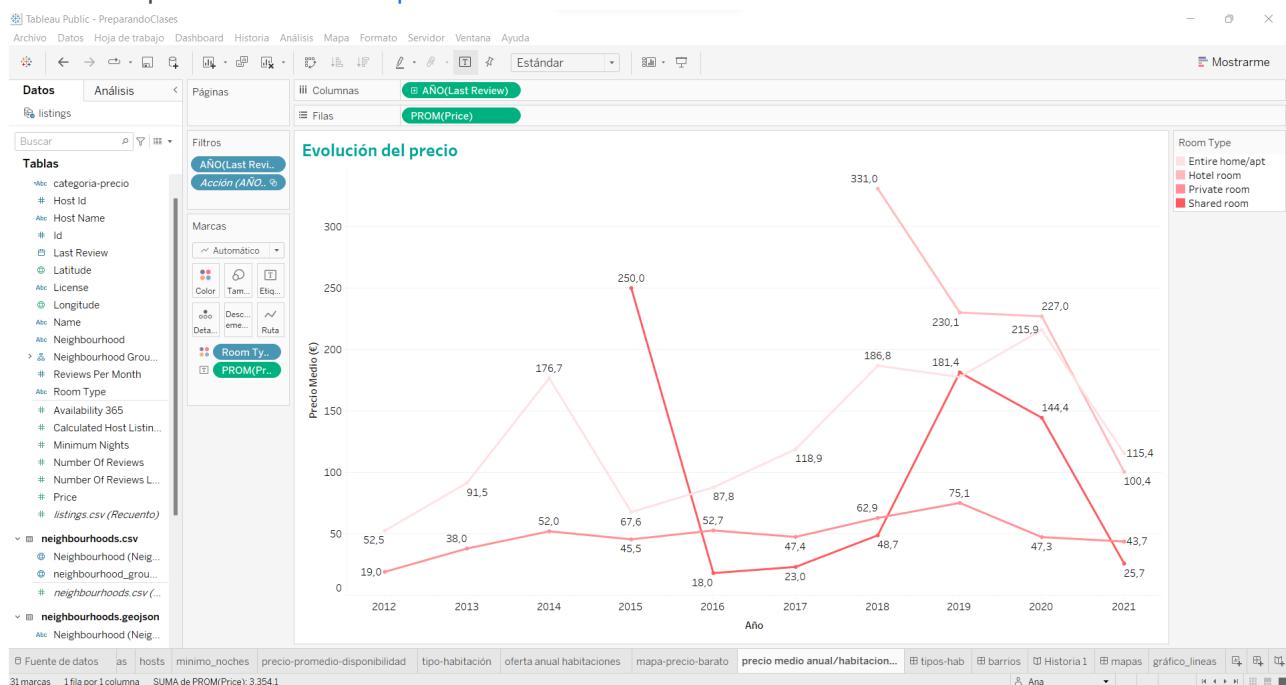


circular

4. **Gráfico de líneas:** es muy útil para comprender tendencias y patrones. Es mejor usar el gráfico de líneas si desea mostrar datos relativos a una variable continua como el tiempo. Las diferentes líneas de colores para diferentes variables en los datos facilitan la comprensión de un gráfico de líneas.

Después del video deberíais tener un gráfico similar a este

El video lo podréis encontrar [aqui](#)

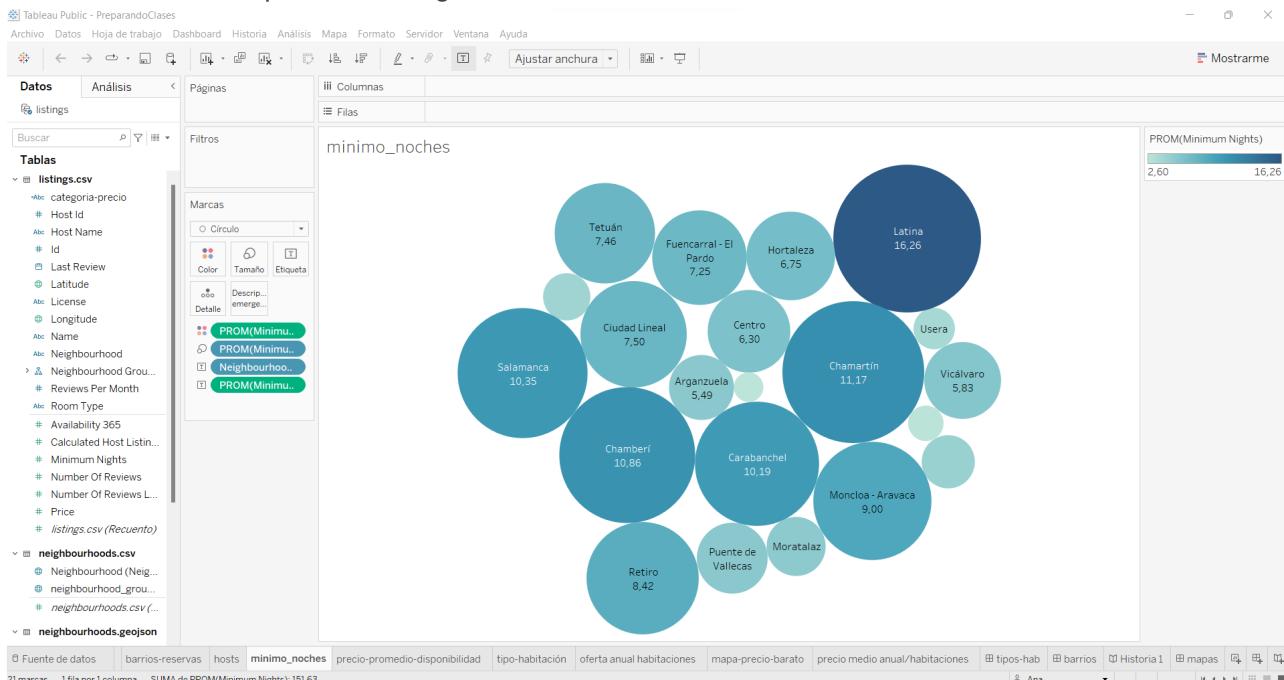


lineas

5. **Gráfico de área:** es similar a un gráfico de líneas en que también visualiza los datos en forma de una línea que es muy útil para comprender tendencias y patrones. Sin embargo, el área debajo de la línea en un gráfico de áreas también está coloreada. Esto se puede usar con múltiples variables en los datos para demostrar las diferencias relativas entre las variables.
6. **Gráfico de dispersión:** los diagramas de dispersión se utilizan para comprender la relación entre dos variables en los datos. También puede encontrar los valores atípicos en sus datos o comprender la distribución general trazando un diagrama de dispersión. Si los datos se mueven de la parte inferior izquierda a la superior derecha, puede haber una correlación positiva entre las dos variables, si los datos se mueven en la dirección opuesta, puede haber una correlación negativa.
7. **Gráfico de burbujas:** se puede utilizar un gráfico de burbujas para mostrar las relaciones entre diferentes medidas y dimensiones. Si bien un gráfico de burbujas no es una visualización completa por sí solo, se puede usar junto con otros tipos de gráficos para agregar más detalles. El tamaño y el color de las burbujas en un gráfico de burbujas denota varias características de los datos en la visualización.

El video explicativo [aquí](#)

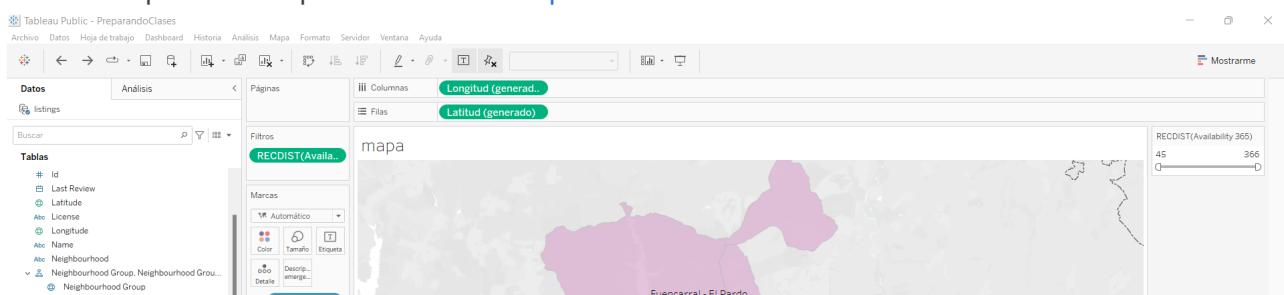
Al final del video, se puede ver un gráfico similar a este .

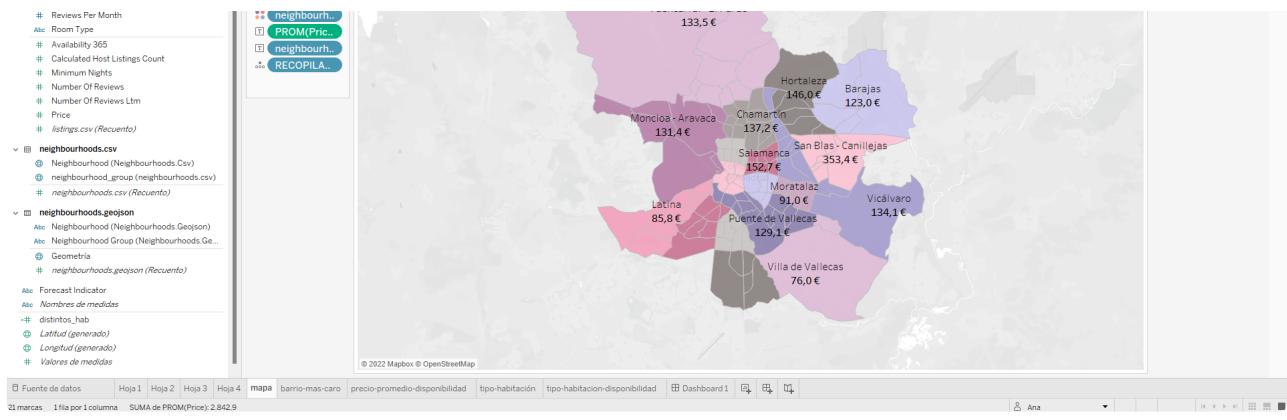


burbuja

8. **Mapa de densidad:** un mapa de densidad visualiza los datos en la parte superior de un mapa de una ubicación geográfica. Es mejor usar este mapa si los datos tienen la geografía como una parte importante con diferentes tonos del mismo color que representan diferentes significados de datos en el mapa. Sin embargo, si desea mostrar puntos precisos de datos, un mapa de densidad no es la mejor idea.

El video explicativo los podréis encontrar [aqui](#)





Imagen

9. **Mapa de calor:** un mapa de calor proporciona la relación entre dos variables en los datos junto con la información de calificación entre estas variables. Esta información de clasificación se muestra normalmente utilizando varios parámetros, como tonos del mismo color, tamaño creciente o decreciente, etc.

En esta lección hemos aprendido los principales tipos de gráficos de Tableau y cómo los podemos hacer desde Tableau. En la siguiente lección aprenderemos como cambiar los colores de los gráficos o aplicar filtros.

Material adicional

En [este link](#) tenéis una pequeña explicación de como es la interfaz principal de las hojas de trabajo de Tableau.

📎
Tableau-II.md
7KB

Binary

Descarga este fichero para tener la lección.

Tableau III - Colores

En esta sesión aprenderemos como cambiar los colores, cuando es mejor usar una paleta de colores u otra y cómo podremos aplicar filtros a nuestras visualizaciones.

1. Color

Los colores que eligamos para nuestras visualizaciones puede parecer banal, pero la realidad es que el impacto que genere nuestra visualización puede asegurar el consumo de la información. La facilidad con la que se cumpla este objetivo depende, en gran medida, del formato de nuestro *dashboard*.

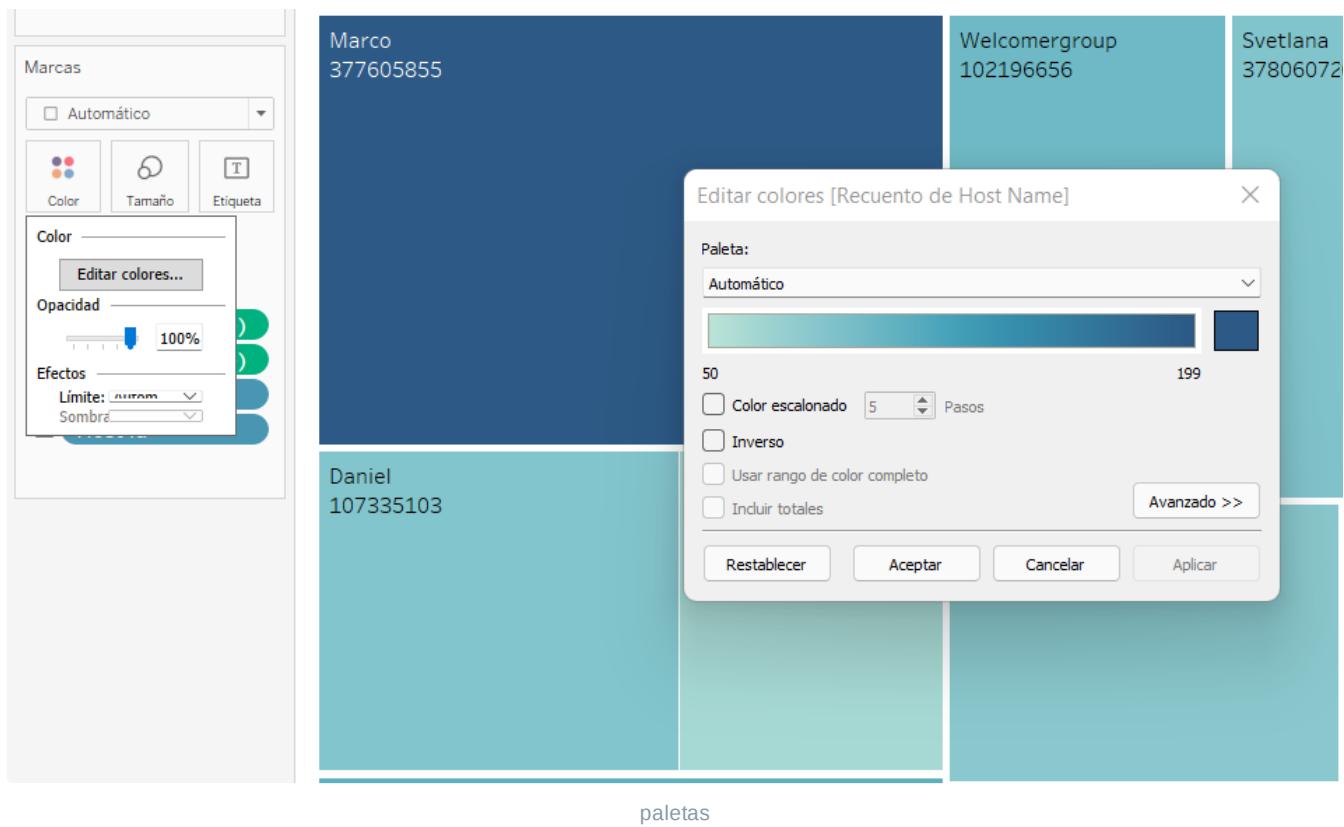
La orientación del análisis visual se puede dirigir de la manera correcta a través del uso del color, entre los expertos dan las siguientes recomendaciones:

- Manejar un uso adecuado en cuanto a la cantidad y combinación de colores facilitará la lectura del análisis y evitará sobrecarga visual para los usuarios, es decir, no debemos trabajar con muchos colores distintos en nuestros *dashboards*, e intentar elegir colores que sean similares entre ellos.
- Mantener consistencia evitando cambiar el significado de los colores en diferentes Dashboards o historias, es decir, si elegimos los colores morado y verde para mujeres y hombres respectivamente deberíamos mantener esos colores a lo largo de toda la visualización cuando queramos usar esas variables a lo largo del *dashboard*.
- Tener en cuenta la deficiencia visual que puedan presentar algunos usuarios (daltonismo)
- Incluir leyendas bajo estrategias visuales para simplificar la lectura del *dashboard*.

El color predeterminado para una vista en Tableau es azul, cambiará cuando arrastremos un campo a la marca de color dependiendo del tipo de dato. Sin embargo, podemos editar el color principal, de la marca en general, lo cual aprenderemos a lo largo de esta lección.

Esta opción para la visualización de datos consiste en la asignación de una serie de colores que identifiquen y dinamicen los datos presentados. Las paletas de color en Tableau también se dividen según sus características, de manera que guíe la comprensión de la información.

Para llevar a cabo el cambio de la paleta de color en Tableau, debes dar clic en «Color» y después en «Editar colores», como se muestra a continuación: En general nos podemos encontrar dos tipos de paletas de colores:



1.1. Tipos de paletas de colores

A partir de esta ventana se desplegarán una gran cantidad de colores y sus tonalidades. En función de como deseemos presentar la información deberemos escoger una paleta u otra, eligiendo la que más se ajuste a nuestros objetivos. Tableau nos ofrece tres tipos de paletas de colores:

- **Paletas de color categóricas:** como su propio nombre indica se recomienda su uso cuando los datos que queremos visualizar sean de tipo categórico, donde cada categoría de nuestra variable a representar se identifique con un color.
- Paletas divergentes: se suelen usar en aquellos momentos donde hay un orden de los valores pero también hay un centro definido. Por ejemplo alturas respecto al nivel del mar(que pueden ser positivas, negativas o cero).
- **Paletas de color secuencial:** es la opción automática dentro de las paletas de color de Tableau. Este tipo de paletas constan de un único color que va desde su tono más claro hasta el más oscuro. Este tipo de paletas se suele usar con información métrica, por ejemplo alturas, precios, temperaturas, etc.
- **Paletas de color divergente:** este tipo de paleta es la más personalizada. Es decir, puede ir de un color a otro según la preferencia de quien cree la visualización. Este tipo de paleta nos permitirá seleccionar los colores, sus tonalidades y la manera en la que se escalarán.

Como otras muchas plataformas de visualización Tableau también nos va a permitir elegir colores que estén fuera de sus colores previamente definidos. En el siguiente video os mostramos como:

Por último, Tableau nos permite personalizar nuestras paletas de color. Esto es un bonus, os dejamos por [aquí](#) un link de la página oficial de Tableau donde nos lo explican.

Aquí tenéis una colección de videos donde cambiamos el color a las gráficas que hemos ido creando:

- [Video 1](#)
- [Video 2](#)
- [Video 3](#)
- [Video 4](#)

2. Filtros

Ayer aprendimos cómo hacer algunas de las gráficas más importantes de Tableau. Pero vimos que en algunos casos nos quedaban gráficas con demasiada información que podía llevar a ser algo difícil de interpretar. En esta parte de la lección aprenderemos como aplicar filtros a nuestras visualizaciones. Además el filtrado de datos es una parte clave del análisis de datos

2.1. Orden de filtrado de las operaciones

Antes de empezar a filtrar datos en Tableau, es importante entender el orden en el que Tableau ejecuta los filtros en las hojas de trabajo.

Tableau realiza acciones en la vista siguiendo un orden concreto; a esto se le llama el **Orden de las operaciones**. Los filtros se ejecutan en el orden siguiente:

- Filtros de extracciones
- Filtros de fuentes de datos
- Filtros de contexto
- Filtros en dimensiones (ya sea en el estante Filtros o en tarjetas de filtros en la vista)
- Filtros en medidas (ya sea en el estante Filtros o en tarjetas de filtros en la vista)

2.2. Tipos de filtros:

2.2.1. Seleccionar mantener o excluir puntos de datos en la vista

Puede filtrar puntos de datos individuales (marcas) o una selección de puntos de datos desde la vista. Por ejemplo, si dispone de un diagrama de dispersión con valores anómalos, puede excluirlos de la vista para poder centrarse mejor en el resto de los datos.

Para filtrar marcas de la vista, seleccione una sola marca (punto de datos) o haga clic y arrastre a la vista para seleccionar varias marcas. En la descripción emergente que aparece, puede:

- Seleccionar Mantener solamente (*Keep only* para la versión en inglés) para mantener las marcas seleccionadas en la vista.
- Seleccionar Excluir (*Exclude* para la versión en inglés) para eliminar las marcas seleccionadas de la vista.

FILTRO SOBRE LA GRAFICA DE LINEAS

2.2.2. Seleccionar encabezados para filtrar datos

También puede seleccionar encabezados para filtrarlos de la vista. Esto lo podremos hacer solo para las representaciones tipo tabla.

Para filtrar filas o columnas de datos enteras de la vista, seleccione el encabezado en la vista. En la descripción emergente que aparece, seleccione **Excluir** o **Mantener** solamente los datos seleccionados.

Cuando seleccionamos el encabezado de una tabla, todos los encabezados del nivel siguiente también se seleccionan. Por ejemplo, la vista que se muestra debajo consta de dos variables colocadas en el estante Filas.

Los encabezados de la fila incluyen los pisos en alquiler para el tipo de vivienda "Private Room" y todos los barrios en los que hay este tipo de alquileres ya que no le hemos aplicado ningún filtro a esa columna.

Cuando se selecciona **Room Type**, todos los miembros del nivel siguiente (en este caso

Neighbourhood) de la jerarquía se seleccionan de manera automática. En este caso, se refiere a los siguientes miembros: estantes para libros, sillas, muebles de oficina y mesas.

2.2.3. Arrastrar dimensiones, medidas y campos de fecha al estante Filtros

Otra manera de crear un filtro es arrastrar un campo directamente del panel Datos al estante Filtros.

2.2.4. Filtrar datos categóricos (dimensiones)

Como vimos en la primera lección, las dimensiones contienen datos categóricos discretos, de manera que filtrar este tipo de campo suele implicar seleccionar valores que incluir o excluir.

Al arrastrar una dimensión desde el panel Datos al estante Filtros en Tableau, aparece el siguiente cuadro de diálogo Filtro:

Filtro [Neighbourhood]

X

General Comodín Condición Límite

Seleccionar de la lista Personalizar lista de valores Usar todo ≡

Escribir texto de búsqueda

- Abrantes
- Acacias
- Adelfas
- Aeropuerto
- Aguilas
- Alameda de Osuna
- Almagro
- Almenara
- Almendrales
- Aluche
- Ambroz

Excluir

Resumen

Campo: [Neighbourhood]
Selección: 128 de 128 valores seleccionados
Comodín: Todo
Condición: Ninguno
Límite: Ninguno

filtro

Veamos paso a paso que significan cada una de las pestañas que vemos:

- General (General en la versión inglesa): usaremos esta pestaña para seleccionar los valores que

- **General** (General en la versión inglesa): usaremos esta pestaña para seleccionar los valores que deseé incluir o excluir.
- **Comodín (solo en Tableau Desktop)** (*Wildcard* en la versión inglesa): usaremos esta opción para definir un patrón de filtrado. Por ejemplo, al filtrar direcciones de correo electrónico, se recomienda incluir solo correos de un dominio específico. Puede definir un filtro de comodín que termine en "@gmail.com" para incluir solo las direcciones de correo electrónico de Google. Nosotras no lo podremos usar ya que estamos trabajando con Tableau Public .
- **Condición** (*Condition* en la versión inglesa): use la pestaña Condición del cuadro de diálogo Filtrar para definir las reglas de filtrado. Por ejemplo, en una vista que muestra el promedio del precio del alquiler de casas, tal vez solo queramos mostrar las casas cuyo precio sea mayor o igual a 25. Puede usar los controles integrados para escribir una condición o bien puede escribir una fórmula personalizada.
- **Principales** (*Top* en la versión inglesa): usaremos esta opción para definir una fórmula que calcule los datos que se incluirán en la vista. Por ejemplo, en una vista que muestra el promedio de tiempo de envío de una colección de productos, puede optar por mostrar solo los 15 productos principales (o los más inferiores) en términos de ventas. En lugar de tener que definir un rango específico para Ventas (por ejemplo, superior a 100 000 \$), puede definir un límite (los 15 primeros) relativo a los demás miembros del campo (productos).

Nota importante: cada pestaña añade definiciones adicionales al filtro. Por ejemplo, podemos optar por excluir valores en la pestaña General y añadir límites en la pestaña Principales. Las selecciones y configuraciones de ambas pestañas se aplican al filtro.

Podremos ver las definiciones de nuestros filtros cuando deseé en **Resumen** (*Sumamry* en la versión en inglés), en la pestaña General.

2.2.5. Filtrar datos cuantitativos (medidas)

Las medidas contienen datos cuantitativos, de manera que filtrar este tipo de campo suele implicar seleccionar un rango de valores que deseé incluir.

Al arrastrar una medida desde el panel Datos al estante Filtros en Tableau Desktop, aparece el siguiente cuadro de diálogo:

¿Cómo desea filtrar en [Number Of Reviews]?

- Todos los valores**
- Suma
- Promedio
- Mediana
- Recuento
- Recuento (Distintos)
- Mínimo
- Máximo
- Desviación estándar
- Desviación estándar (población)
- Varianza
- Varianza (población)
- Atributo

Siguiente >

Cancelar

filtro

Una vez en esta pestaña debemos seleccionar cómo queremos agregar el campo y, a continuación, haremos clic en **Siguiente**.

El siguiente cuadro de diálogo nos proporciona la opción de crear cuatro tipos de filtros cuantitativos:

- **Intervalo de valores:** seleccionaremos esta opción para especificar los valores mínimo y máximo del rango para incluir en la vista. **Los valores que especificamos se incluyen en el rango.**
- **Mínimo:** la usaremos para incluir todos los valores mayores o iguales a un valor mínimo especificado. Este tipo de filtro es útil cuando los datos cambian a menudo, por lo que especificar un límite superior puede no ser posible.
- **Máximo:** la usaremos para incluir todos los valores menores o iguales a un valor máximo especificado. Este tipo de filtro es útil cuando los datos cambian a menudo, por lo que especificar un límite inferior puede no ser posible.
- **Especial:** lo usaremos para filtrar los valores nulos. Incluya solo valores nulos, valores no nulos o todos los valores.

Nota: Si tenemos una fuente de datos muy grande, filtrar medidas puede afectar al rendimiento considerablemente, es decir, puede hacer que el procesado y visualización de la vista sea más lenta.

2.2.6. Filtrar fechas

Al arrastrar un campo de fecha desde el panel Datos al estante Filtros en Tableau, aparece el siguiente cuadro de diálogo Filtrar campo:

Filtrar campo [Last Review]

X

¿Cómo desea filtrar en [Last Review]?

- Fecha relativa
- Rango de fechas
- Años
- Trimestres
- Meses
- Días
- Números de semanas
- Días de semana
- Mes / Año
- Día / Mes / Año
- Fechas individuales
- Recuento
- Recuento (Distintos)
- Mínimo
- Máximo
- Atributo

Siguiente >

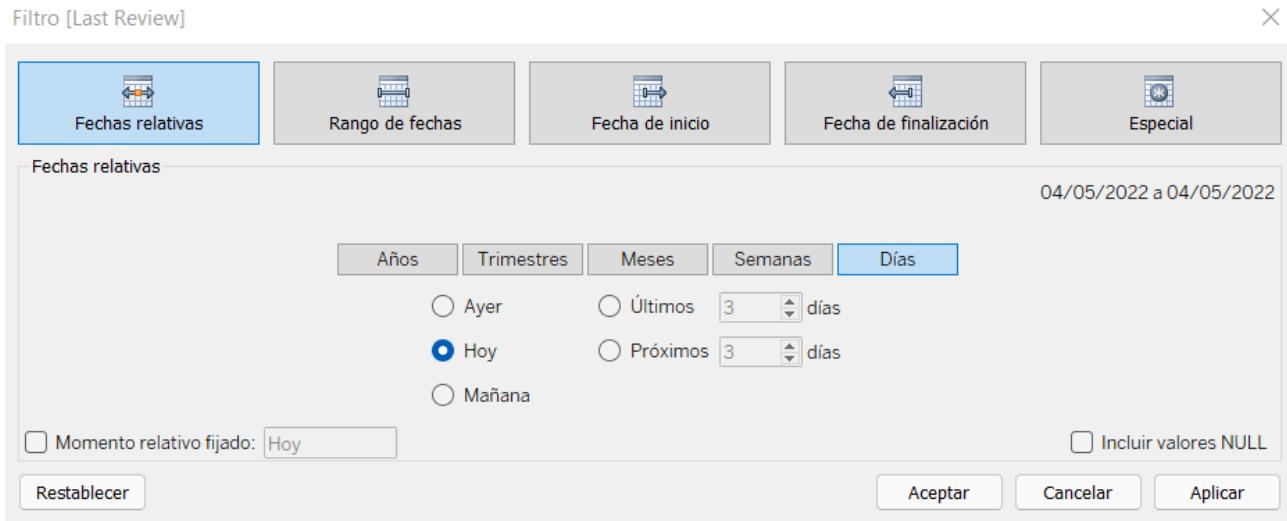
Cancelar

filtro

De nuevo, tenemos varias opciones para aplicar filtros sobre fechas:

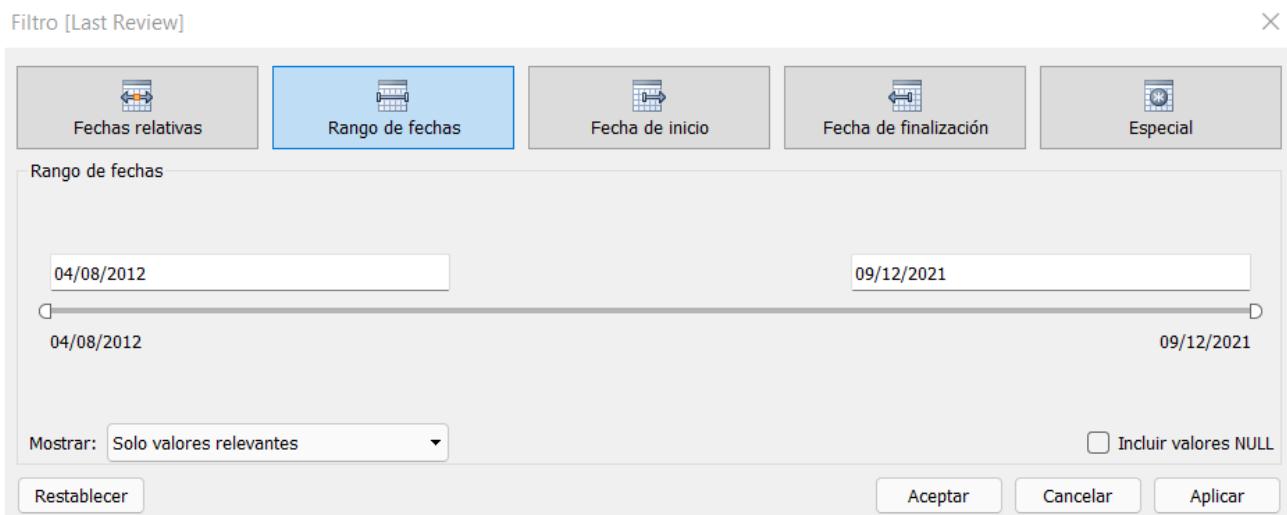
- **Filtrar fechas relativas** (*Relative Date* en inglés): lo usaremos para definir un intervalo de fechas que

- **Filtrar fechas relativas** (Relative Date en inglés), lo usaremos para definir un intervalo de fechas que se actualice según la fecha y la hora en que se abra la vista. Imaginemos que queremos filtrar los alquileres de los últimos 30 días desde el día de hoy.



fecha_relativa

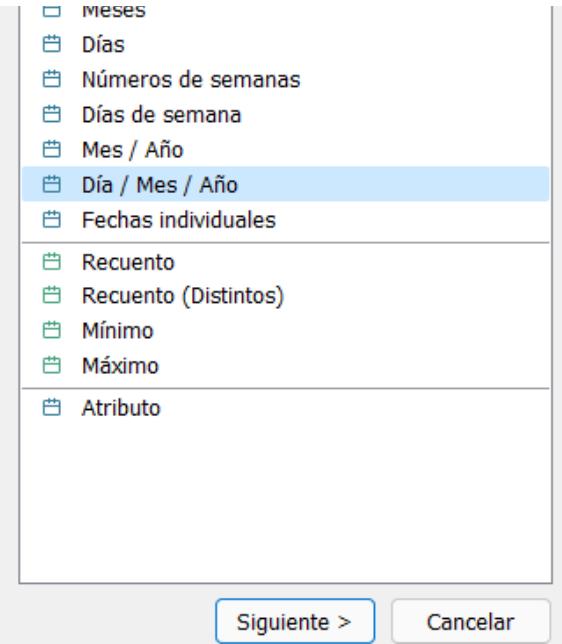
- **Filtrar un rango de fechas** (Ranges of Dates en inglés): seleccione Intervalo de fechas para definir un intervalo de fechas fijo para filtrarlo. Por ejemplo, es posible que quiera ver todos los pedidos hechos entre el 1 de marzo de 2009 y el 12 de junio de 2009.



filtro_rango

- **Filtrar fechas discretas**: seleccionaremos un valor de fecha discreta (Año, mesm días, número de semana, etc.) en el cuadro de diálogo si deseamos incluir niveles de datos completos. Por ejemplo, si seleccionamos Trimestres, podemos optar por filtrar trimestres concretos (por ejemplo, T1, T2, T3 o T4) de la vista, independientemente del año.
- **Fecha más reciente predefinida**: si queremos asegurarnos de que en el filtro solo se selecciona la fecha más reciente de una fuente de datos al abrir o compartir el libro de trabajo, seleccione una fecha discreta, como Mes/Día/Año o Fechas individuales y, luego, en la pestaña General, seleccione Filtrar valor de fecha más reciente al abrir el libro de trabajo.





fecha_mas_reciente

Se nos abrirá la siguiente ventana, donde seleccionaremos la fecha que queremos:

Filtro [Mes, Día, Año de Last Review]

General Condición Límite

Seleccionar de la lista Personalizar lista de valores Usar todo

Escribir texto de búsqueda

- NULL
- 4 de agosto de 2012
- 14 de septiembre de 2012
- 23 de septiembre de 2012
- 3 de enero de 2013
- 6 de mayo de 2013
- 18 de mayo de 2013
- 21 de mayo de 2013
- 31 de mayo de 2013
- 28 de julio de 2013
- 19 de septiembre de 2013

Todo Ninguno Excluir

Resumen

Campo: [Mes, Día, Año de Last Review]
 Selección: 0 de 1624 valores seleccionados
 Comodín: Todo
 Condición: Ninguno
 Límite: Ninguno

Filtrar el valor de fecha más reciente cuando se abre el libro de trabajo

Restablecer Aceptar Cancelar Aplicar

fecha_mas_reciente_seleccionada

2.2.7. Filtrar cálculos de tablas

Para hacer este tipo de filtros necesitaremos crear **Campos Calculados**. Como todavía no hemos aprendido no os los introduciremos todavía, pero en la siguiente lección veremos que son estos campos y como podemos filtrarlos.

3. Mostrar filtros interactivos en la vista

Hasta ahora hemos aprendido que tipos de filtros podemos usar en Tableau. Ahora vamos a aprender como podremos mostrar estos filtros en nuestra hoja de trabajo. Esto nos va a permitir mostrar u ocultar los filtros de la vista rápidamente.

3.1. Para mostrar un filtro en la vista:

En la vista, haga clic en el menú desplegable del campo y podemos seleccionar **Mostrar filtro** (*Show Filter* en inglés).

El campo se añade automáticamente al estante Filtros (si no se está filtrando ya) y aparece una tarjeta de filtro en la vista. Interactúe con la tarjeta para filtrar los datos

Podemos acceder a otras opciones haciendo clic en el menú desplegable, en la esquina superior derecha de la tarjeta del filtro en la vista.

Algunas opciones están disponibles para todos los tipos de filtros y otras dependen de si se filtra un campo de categoría (dimensión) o un campo cuantitativo (medida).

Algunas de las opciones disponibles en este desplegable son:

- **Editar filtro:** esta opción abre el cuadro de diálogo principal **Filtrar** para que pueda refinar el filtro

- **Editar filtro:** Esta opción abre el cuadro de diálogo principal para que pueda editar el filtro añadiendo condiciones y límites como las que hemos ido aprendiendo hasta ahora.
- **Eliminar filtro:** elimina el filtro del estante Filtros y elimina la tarjeta de filtro de la vista.
- **Aplicar a hojas de trabajo:** Tableau nos permite crear un filtro en una hoja de trabajo y que se pueda aplicar en otras creadas anteriormente.
- **Formatear filtros (solo para Tableau Desktop):** nos permite personalizar la fuente y los colores de todas las tarjetas de filtros de la vista.
- **Solo valores relevantes:** especifica los valores que se mostrarán en el filtro. Si selecciona esta opción, se consideran otros filtros y se muestran solo los valores que pasan estos filtros. Por ejemplo, un filtro en Estado solo mostrará los estados orientales cuando se haya establecido un filtro en Región.
- **Todos los valores de la base de datos:** especifica los valores que se mostrarán en el filtro. Si selecciona esta opción, se muestran todos los valores de la base de datos, independientemente de los otros filtros de la vista.
- **Todos los valores en contexto (solo para Tableau Desktop):** si uno de los filtros de la vista es un filtro de contexto, seleccione esta opción en otro filtro para mostrar solo los valores que pasan por el filtro de contexto. Para obtener más información, consulte Mejorar el rendimiento de la vista con filtros de contexto.
- **Ocultar tarjeta :** oculta la tarjeta del filtro, pero no elimina el filtro del estante Filtros.

Os recomendamos ver [este](#) video donde os explicamos algunos ejemplos de filtros sobre nuestras gráficas.

Y hasta aquí la lección de hoy donde hemos aprendido la importancia de la elección de los colores en nuestras visualizaciones y como podemos filtrar datos en Tableau.



Tableau-III-Colores.md 17KB
Binary

Descarga este fichero para tener la lección.

Tableau IV - Campos Calculados I

En la lección de hoy aprenderemos que son los campos calculados y como los podemos crear en Tableau.

En ocasiones nos podemos encontrar que en nuestros datos no están incluidos campos que necesitamos para responder a las preguntas. Desde Tableau podremos crear esos campos usando cálculos y luego guardarlos como parte de nuestra fuente de datos. Esto es lo que llamamos **campos calculados**.

1.2. Tipos de campos calculados

El tipo de cálculo que usaremos dependerá de nuestras necesidades y de la pregunta que queremos responder. En este contexto, en Tableau podemos encontrar 3 tipos principales de cálculos

- **Cálculos básicos:** los cálculos básicos le permiten transformar los valores o miembros a nivel de fuente de datos (un cálculo a nivel de fila) o a nivel de detalle de visualización (un cálculo agregado).
- **Expresiones de nivel de detalle (LOD):** al igual que los cálculos básicos, los cálculos de LOD nos permiten calcular valores a nivel de fuente de datos y a nivel de visualización. Sin embargo, los cálculos de LOD nos proporcionan aún más control sobre el nivel de granularidad que desea calcular. Se pueden efectuar a un nivel más granular (INCLUDE), a un nivel menos granular (EXCLUDE) o a un nivel completamente independiente (FIXED) con respecto a la granularidad de la visualización.
- **Cálculos de tabla:** los cálculos de tabla le permiten transformar valores solamente a nivel de detalle de visualización.

1.3. Funciones de Tableau básica

Tableau nos permite hacer muchos tipos de operaciones para crear los campos calculados. Los principales tipos de funciones que podemos encontrarnos son:

1.3.1. Funciones numéricas

Las funciones numéricas nos permiten realizar cálculos en los valores de datos de los campos. Este tipo de operaciones solo se podrán usar en campos de tipo numéricos. Imaginemos que tenemos una columna que es precio, y vemos que tiene valores negativos, eso no podría ser posible. Podríamos usar la función ABS para obtener el valor absoluto de los valores negativos, mientras que los positivos se mantendrán igual.

El cálculo se podría parecer a lo siguiente:

```
ABS[nombre_columna]
```

Algunas de las funciones más importantes son:

Función	Sintaxis	Descripción
CEILING	CEILING([nombre_col])	Redondea un número al entero más cercano de valor igual o superior.
DIV	DIV([col1], [col2])	Indica la parte entera de una operación de división en la que entero1 se divide entre entero2
FLOOR	FLOOR([nombre_col])	Redondea un número al entero más cercano de valor igual o inferior
MAX	MAX([col1],[col2])	Indica el máximo de dos argumentos, los cuales deben ser del mismo tipo. Indica Null uno u otro argumento es Null
MIN	MIN([col1],[col2])	Indica el mínimo de dos argumentos, los cuales deben ser del mismo tipo. Indica Null uno u otro argumento es Null
ROUND	ROUND([nombre_col], decimales)	Redondea los números a una cantidad de dígitos especificada. El argumento decimals especifica la cantidad de puntos decimales que queremos. Si se omite decimal number se redondea al entero más cercano.
SQRT	SQRT([nombre_col])	Indica la raíz cuadrada de un número
SQUARE	SQUARE([nombre_col])	Indica el cuadrado de un número.

Aquí otros tipo de funciones numéricas.

Además, tenéis un video explicando como hacer un video de este tipo [aqui](#)

1.3.2. Funciones de cadena

Las funciones de cadena nos permiten manipular datos de tipo texto.

Por ejemplo, imaginemos que tenemos un campo donde tenemos la ciudad y la comunidad autónoma. Podríamos separar uno de los elementos en una columna separada.

El cálculo en este caso podría ser:

```
-- suponiendo que los valores de nuestras celdas sean  
Riopar, Albacete  
  
SPLIT([nombre_col], ',', 2) -- suponiendo que estén separados por comas. Además debemos espec...
```

Algunas de las funciones más importantes son:

Función	Sintaxis	Descripción
CONTAINS	CONTAINS([nombre_col], "el_substring_que_buscamos") = true	Indica true si la cadena dada contiene la subcadena especificada
LEN	LEN([nombre_col])	Indica la longitud de la cadena
LOWER	LOWER([nombre_col])	Indica la string, con todos los caracteres en minúscula
REPLACE	REPLACE(string, substring, replacement)	Busca <i>string</i> para <i>substring</i> y lo reemplaza con <i>replacement</i> . Si no se encuentra <i>substring</i> , se elimina el cambio a la cadena
SPLIT	SPLIT([nombre_col]), delimitador, posición)	Indica una subcadena a partir de una cadena (usa un carácter delimitador para dividir la cadena, igual que en Python)
UPPER	UPPER([nombre_col])	Indica la cadena, con todos los caracteres en mayúscula

Otras funciones de strings en [este link](#).

1.3.3. Funciones de fecha

Nos van a permitir manipular fechas en nuestra fuente de datos. Imaginemos que tenemos una columna de fechas con año, mes y día. Podríamos ser capaces desacar la fecha de inicio de trimestre para cualquier valor de fecha que tengamos. Para eso tendremos que usar la función `DATETRUNC`, su sintaxis sería la siguiente:

```

DATETRUNC("quarter", [nombre_col]) -- en este caso nombre_col tendrá que ser una columna de tipo fecha

-- ¿Qué nos devolverá esto?

-- Si la fecha es 21/2/2022, la función nos devolverá 1/1/2022 para indicar que el primer trimestre

-- Si la fecha es 07/07/2022, la función nos devolverá 1/7/2022 para indicar que el tercer trimestre

```

Algunas de las funciones más importantes para trabajar con fechas son:

Función	Sintaxis	Descripción
DATEDIFF	DATEDIFF(date_part, [col_fecha_1], [col_fecha_2], [inicio_semana])	Indica la diferencia entre date1 y date2 que se expresa en unidades de date_part. El parámetro inicio_semana es optativo y puede ser desde lunes a domingo, indicaremos en qué día empiezan nuestras semanas. En el parámetro date_part tendremos que pasar la escala, es decir, días, mes, año... (puede que tengamos que meterlo en inglés)
DAY	DAY([nombre_col])	Indica el día de la fecha dada como un entero
MAKEDATE	MAKEDATE([col_año], [col_mes], [col_dia])	Indica un valor de fecha construido a partir del año, del mes y de la fecha especificado
MONTH	MONTH([nombre_col])	Indica el mes de la fecha dada como un entero
WEEK	WEEK([nombre_col])	Indica la semana especificada como un número entero
YEAR	YEAR([nombre_col])	Indica el año de la fecha dada como un entero

De nuevo, más opciones en [este link](#).

1.3.4. Funciones lógicas

Al igual que en Python, estas funciones nos van a permitir chequear si una condición que le pasemos es verdadera o falsa. Podríamos querer saber si los precios de Airbnb de cada barrio están por encima de determinado umbral o no. Si vieramos esto con código en Tableau podríamos escribir:

```
SUM([nombre_col]) > umbral -- donde el umbral será un valor que nosotras especifiquemos.
```

Veamos las funciones lógicas más importantes de Tableau:

Función	Sintaxis	Descripción	Ejemplo
IN	expr1 IN expr2	Devuelve TRUE si cualquier valor coincide con cualquier valor	<code>SUM([col1]) IN (10, 500, 320) OR [col1] IN [col2]</code>
IF	<code>IF expr1 THEN pasa_algo ELSEIF exp2 THEN "pasa_otra_cosa" ELSE pasa_otra_cosa END</code>	Es como el <code>if</code> de Python. Prueba una serie de expresiones que indican el valor.	<code>IF [Price] > 100 THEN "caro" ELSEIF [Price]=100 THEN "medio" ELSE "barato" END</code> . En este caso lo que estamos haciendo es trabajar sobre una columna que se llama precio. Funciona un poco como en SQL, si el valor de la celda es mayor que 100 será caro, si es igual a 100 será medio y si no cumple ninguna de estas condiciones será barato. Con esto estaremos creando una columna nueva con estas tres categorías
NOT	<code>IF NOT expresion THEN pasan_cosas END</code>	Realiza una negación lógica en una expresión	<code>IF NOT [price]>0 THEN "Gratis" END</code> . Con esta expresión estamos indicando que si el precio no es mayor que cero, nos categorice ese registro como gratis
	<code>IF expr1 OR expr2</code>	Es como nuestro OR	<code>IF [precio] < 0 OR [precio] = 0 THEN "Necesitamos revisar estos datos" ELSE "el precio esta</code>

OR	THEN pasan cosas END	de Python. Chequea si se cumple una condición u otra	bien" END . Con esto lo que estamos diciendo es que si el precio es menor o igual a 0, esos valore hay que revisarlos, por el contrario nuestra nueva columna tendrá el valor de "el precio esta bien"
AND	IF expr1 AND expr2 THEN PASAN COSAS END	Como nuestro "and" de Python para chequear que se cumplan dos condiciones a la vez	IF [precio] == AND [barrio] = "Salamanca" THE "Tenemos casa gratis" ELSE "habrá que pagar" END . En este caso se tendrán que cumplir las dos condiciones

Video explicativo de como crear un campo de este tipo [aquí](#)

Para más operaciones lógicas, seguid [este link](#).

1.3.5. Funciones de agregación

Las funciones agregadas nos van a permitir resumir o cambiar la granularidad de los datos. Pero... ¿esto qué significa exactamente ?

Imaginemos que queremos saber el número de personas que han usado por año los servicios de Airbnb en Madrid. En nuestro caso, tenemos una columna que se llama `Name` que nos da el nombre de la persona que alquila. Por otro lado, en Tableau tenemos la función `COUNTD` que nos cuenta el número de elementos distintos que tenemos, como el `DISTINCT` de SQL, con esto tendremos una nueva columna en nuestros datos. Si luego queremos saber el número de personas por año, lo único que tendremos que hacer será incluir el año en la visualización y ya tendremos el conteo por año.

La sintaxis sería esta:

```
COUNTD([nombre_col])
```

Antes de pasar a ver las funciones más usadas hablemos de las **Reglas de los cálculos agregados**

- En el caso de cualquier cálculo de agregación, no puede combinar un valor agregado con un valor desasociado. Por ejemplo, `SUM(Price)*[Name]` no es una expresión válida porque `SUM(Price)` está agregado y `Items` no. Sin embargo, tanto `SUM(Price*Name)` como `SUM(Price)*SUM(Name)` son válidas.
- Los números en una expresión actúan como valores agregados o desasociados, según corresponda. Por ejemplo, tanto `SUM(Price*7)` como `SUM(Price)*7` son expresiones válidas.
- Todas las funciones se pueden evaluar según valores agregados. Sin embargo, los argumentos ante cualquier función dada deben ser o todos agregados o todos desasociados. Por ejemplo, `MAX(SUM(Sales),Profit)` no es una expresión válida porque `Sales` está agregado y `Profit` no. No obstante, `MAX(SUM(Sales),SUM(Profit))` sí es una expresión válida.
- El resultado de un cálculo de agregación siempre es una medida.

Una vez vistas las reglas, vemos las funciones de agregación más usadas en Tableau son:

Función	Sintaxis	Descripción
SUM	SUM([nombre_col])	Indica la suma de todos los valores de la expresión. SUM solo puede usar solo con campos numéricos. Se ignoran los valores nulos
MIN	MIN([nombre_col])	Indica el mínimo de una expresión en todos los registros. Si la expresión es un <i>string</i> , esta función devuelve el primero por orden alfabético.
MAX	MAX([nombre_col])	Indica el máximo de una expresión en todos los registros. Si el valor es un <i>string</i> , esta función devuelve el último ordenados alfabéticamente
AVG	AVG([nombre_col])	Indica el promedio de todos los valores de la expresión. Solo válido para campos numéricos
VAR	VAR([nombre_col])	Nos devuelve la varianza de la columna
STDEV	STDEV([nombre_col])	Nos devuele la desviación estándar
COUNT	COUNT([nombre_col])	Indica el número de elementos que hay
COUNTD	COUNTD([nombre_col])	Indica el número de elementos distintos que hay

Hasta aquí la lección de hoy. En la siguiente lección seguiremos aprendiendo otro tipo de funciones de Tableau.



Tableau-IV-CC-I.md 12KB
Binary

Descarga este fichero para tener la lección.

Tableau V - Campos Calculados II

En la lección de hoy seguiremos aprendiendo más funciones de Tableau para crear campos calculados.

En la lección de ayer vimos como Tableau tiene muchísimas opciones para realizar campos calculados. Vimos como podíamos hacer operaciones con números, *strings*, fechas, o incluso operaciones lógicas y de agrupación de una forma muy similar a como lo haríamos en Python.

Hoy seguiremos aprendiendo funciones más complejas de Tableau.

1. Campos calculados

Recordemos que los campos calculados son una forma de agregar información a una tabla a partir de datos que si que existen en nuestra fuente de datos.

1.1. Funciones avanzadas

En la lección de hoy profundizaremos un poco más en las funciones avanzadas de Tableau.

NOTA Estas son funciones avanzadas de Tableau y no es necesario que las implementéis en vuestros proyectos

1.1.1. Funciones de usuario

Las funciones de usuario sirven para crear filtros de usuario o filtros de seguridad de nivel de fila que afectan a las visualizaciones publicadas en Tableau Server o Tableau Online de forma que solo determinadas personas puedan ver la visualización.

Imaginemos que en nuestra empresa nos piden que creamos un *dashboard* donde podremos ver los rendimientos de todos los empleados y después publicamos esa información en Tableau Server o Tableau Online. Sin estas funciones de usuarios, todos los empleados podrían ver los rendimientos de sus compañeros, cuando en realidad nuestros jefes no quieren que esto ocurra. Con este tipo de funciones conseguiremos que los empleados solo puedan ver sus rendimientos y no los de sus compañeros.

1.1.2. Funciones espaciales

Las funciones espaciales nos permiten realizar análisis espacial avanzado y combinar archivos espaciales con datos en otros formatos, como archivos de texto u hojas de cálculo. Por ejemplo, puede tener un archivo espacial como nuestro archivo de `neighbourhoods.geojson` y un archivo de texto como por ejemplo un archivo donde tengamos coordenadas de bares. Podemos utilizar un cálculo espacial al crear la fuente de datos para unir estos archivos y analizar qué casas tienen bares cerca

Otro ejemplo podría ser crear líneas que conecten dos puntos en mapas de origen y destino. Por ejemplo, imaginemos que tenemos información sobre los movimientos de personas entre sus hogares y sus centros de trabajo. Usando esta herramienta, podríamos ver las rutas de las personas que van de hogar a trabajo y viceversa.

La lista completa de las funciones de Tableau la podréis encontrar [aquí](#)

Otros tipos de funciones que podemos encontrar en Tableau son:

- Funciones de cálculo de tablas
- Funciones adicionales
- Funciones de paso

Y hasta aquí la lección de hoy !



Tableau-V-CC-II.md 3KB

Binary

Descarga este fichero para tener la lección.

Tableau VI - Dashboards I

En esta lección aprenderemos como crear Dashboards con todas las gráficas que nos hemos ido creando hasta ahora. Es decir, combinaremos todas nuestras gráficas en una única visualización.

1. ¿Qué es un *dashboard*?

Un *dashboard* es una herramienta de gestión de la información que monitoriza, analiza y muestra de manera visual los indicadores clave de desempeño (KPI), métricas y datos fundamentales para hacer un seguimiento del estado de una empresa, un departamento, una campaña o un proceso específico.

Podemos pensar en el *dashboard* como una especie de "resumen" que recopila datos de diferentes fuentes en un solo sitio y los presenta de manera digerible para que lo más importante salte a la vista. Estas son algunas de las características que debe tener este centro de control:

- **Personalizado:** Un dashboard debe contener únicamente los KPI que sean relevantes para el departamento, campaña o proceso que nos ocupa. Para orientarlo, podemos pensar en las preguntas principales a las que queremos responder. Por ejemplo, cuáles son las principales fuentes de tráfico a nuestra web, cómo está funcionando nuestro embudo de ventas o cuáles son los 5 productos que nos generan más ingresos.
- **Visual:** La idea de un dashboard es que podamos obtener la información que buscamos a golpe de vista. Por ello, los datos se presentan en forma de gráficos y debemos contar con indicadores rápidos a través de claves de color, flechas hacia arriba o abajo o cifras destacadas, por ejemplo.
- **Práctico:** La función principal de un dashboard siempre debe ser orientar las acciones de nuestro equipo. Por tanto, debe facilitarnos la información necesaria para que podamos saber cuáles son los siguientes pasos a seguir para mejorar los resultados.
- **En tiempo real:** A día de hoy, las acciones de marketing digital evolucionan con gran rapidez y aprovechar el momento clave es esencial. Por eso, la información debería estar actualizada al momento en todas las fuentes y mostrarse en el dashboard en tiempo real.

2. Consejos para crear *dashboards* eficaces

- Lo primero que nos tenemos que preguntar cuando hacemos un *dashboard* es: ¿Qué tipo de información queremos visualizar? ¿Cuál es nuestro objetivo?

En este contexto, conocer nuestros objetivos y quienes van a ser los usuarios que van a visualizar nuestro *dashboard* es una de las primeras cosas que debemos hacer. Llevando este ejemplo al ridículo, podemos entender que no es lo mismo explicarle a nuestros amigos como se ha movido el sector del Airbnb en Madrid que contárselo a nuestros abuelos. De la misma forma, no será lo mismo contárselo al cliente final que a nuestro equipo de desarrollo.

Así, además de saber que es lo que estamos haciendo, es importante plantearse las siguientes preguntas:

- ¿Conoce nuestro público este tema lo suficientemente bien o les resultará nuevo?
- En caso de que les resulte nuevo, ¿qué tipo de pistas o conocimientos necesitaremos explicar o incluir en el *dashboard* para que puedan entenderlo todo?
- Debemos aprovechar el punto más visto en nuestro *dashboard*.
El punto más visto de una visualización, ya sea una página web, un *dashboard* o póster es la **esquina superior izquierda**. Es por este motivo, que una vez que sepamos cuál es la visualización más importante de nuestro *dashboard*, debemos colocarlo en esa posición.
- Crear en el tamaño de visualización final.
Como hemos visto en el video anterior, cuando iniciamos un *dashboard* su tamaño por defecto es fijo. Si decidimos elegir esta opción debemos asegurarnos que el tamaño que seleccionamos es el tamaño en el que se verá. Entendemos que no es lo mismo ver un *dashboard* en una pantalla de ordenador pequeño que en una pantalla adicional con mayores dimensiones.
Otra opción es seleccionar el tamaño como `automático`, en este caso Tableau adaptará automáticamente las dimensiones del *dashboard* a las dimensiones de la pantalla. Esto significa que, si diseña un dashboard con 1300 x 700 píxeles, Tableau lo redimensionará para pantallas más pequeñas y, en ocasiones, esto provocará vistas o barras de desplazamiento apretujadas. La función de cambio de tamaño `Intervalo` es útil para evitar este problema.
- Limitar el número de vistas.
O lo que es lo mismo, menos es más. Como normal general, se suele recomendar no incluir más de dos o tres vistas (gráficas) por *dashboard*.
Si añadimos demasiadas vistas nos exponemos a que las gráficas en el *dashboard* se vean demasiado pequeñas o que haya demasiada información.
- Añadir interactividad para estimular la exploración.
 - Mostrar filtros, como hemos aprendido los filtros permiten interactuar al usuario para especificar los datos que quieren ver.
 - Usar el resaltador como aprendimos en la lección 3.

Material adicional

- En [este](#) link tenéis un video de Platzi donde hablan sobre buenas prácticas y errores comunes a la hora de crear un *dashboard*.



Tableau-VI-Dashboards-I.md 5KB

Binary

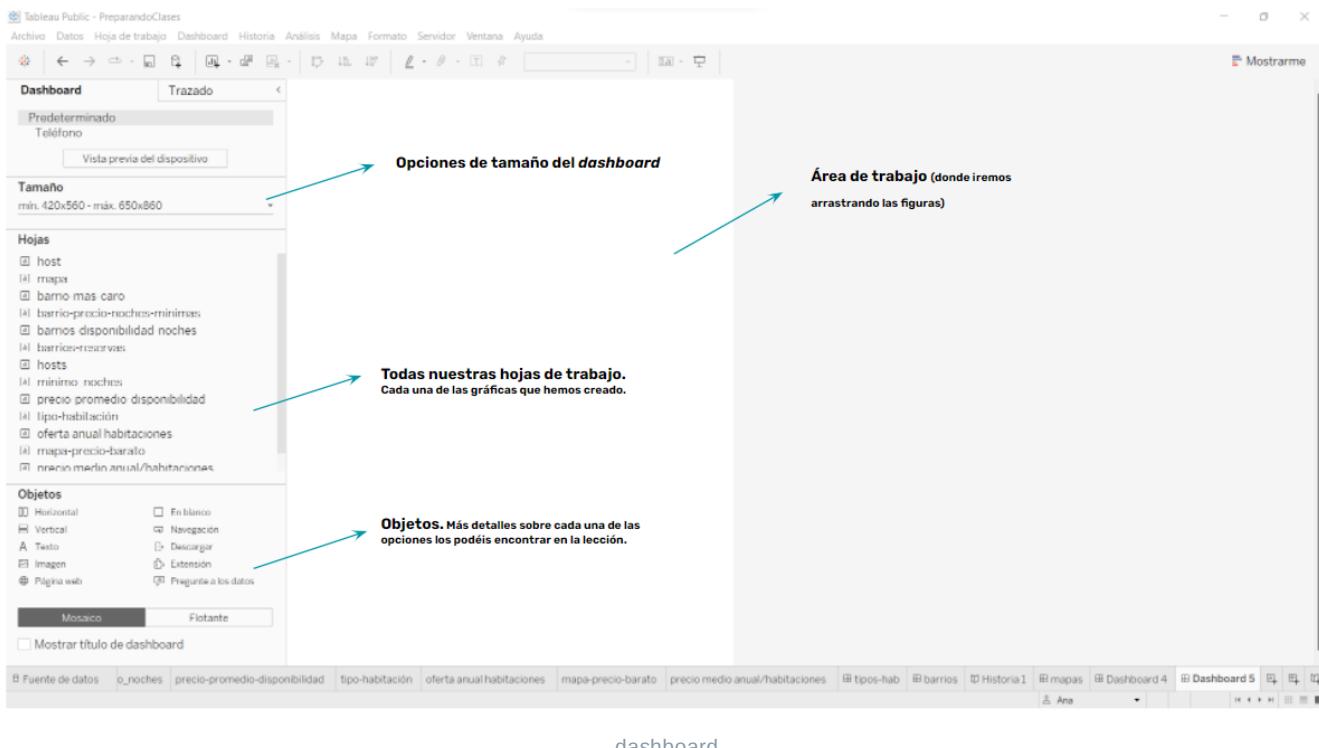
Descarga este fichero para tener la lección.

Tableau VII - Dashboards II

En la lección crearemos un par de dashboards que nos permitirán visualizar los datos que hemos estado trabajando hasta ahora.

1. Interfaz de la pestaña de *dashboard*

Las principales partes de una pestaña de un *dashboard* son:



Sin embargo, para más detalle sobre esta interfaz, os recomendamos ver [este video](#).

Objetos

Además de hojas, podemos añadir **objetos** en nuestro *dashboard* que aumentan el atractivo visual y la interactividad. Veamos más en detalle cada uno de ellos:

- Los **objetos horizontales y verticales** proporcionan contenedores de trazados que permiten agrupar objetos relacionados y ajustar el cambio de tamaño del dashboard cuando los usuarios interactúan con estos objetos.
- Los **objetos de Texto** pueden proporcionar encabezados, explicaciones y otra información.
- Los **objetos de imagen** añaden atractivo visual a un dashboard y puede vincularlos a direcciones URL de destino específicas.
- Los **objetos de página web** muestran páginas de destino en el contexto de nuestro dashboard.
- Los **objetos en blanco** nos ayudan a ajustar el espacio entre elementos del *dashboard*.
- Los **objetos de Navegación** permiten navegar de un dashboard a otro o a otras hojas o historias. Puede mostrar un texto o una imagen para indicar el destino del botón a sus usuarios, especificar los colores personalizados de los bordes y del fondo, y proporcionar información sobre las herramientas. Descargar objetos permite a los usuarios crear rápidamente un archivo PDF, una diapositiva de PowerPoint o una imagen PNG de un dashboard completo o una tabulación cruzada de las hojas seleccionadas. Las opciones de formato son similares a las de los objetos de navegación. Nota: Solo se puede realizar la descarga de la tabulación cruzada después de publicarla en Tableau Online o Tableau Server.
- Los **objetos de extensión** permiten añadir funciones exclusivas a los dashboards o integrarlos con aplicaciones fuera de Tableau.

En [este](#) link de Tableau Public podréis ver *dashboard* de otros usuarios.

Nota Probablemente necesitemos registranos para poder acceder a verlos.

2. Creando nuestro primer *dashboard*

Para montar nuestro primer *dashboard*, os recomendamos que veáis [este](#) video y [este](#), el cuál podréis ir siguiendo al mismo tiempo y crear vuestro primer *dashboard* también.



Tableau-VII-Dashboards-II.md 3KB
Binary

Descarga este fichero para tener la lección.

Tableau VIII - Historias

En esta lección aprenderemos a como crear historias en Tableau.

1. Qué es una historia y para que las usamos

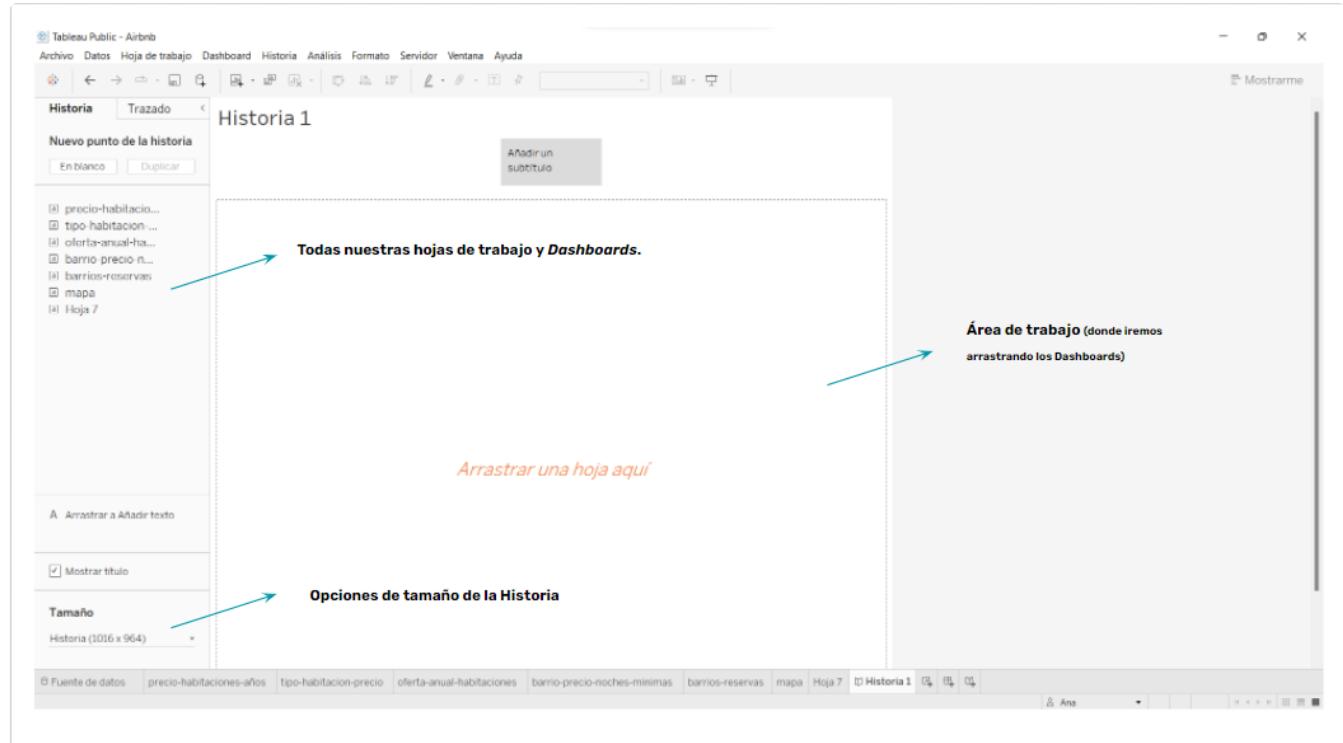
En Tableau, una historia es una secuencia de visualizaciones que se usan en conjunto para transmitir información. Podemos crear historias para contar una narración de datos, proporcionar contexto, mostrar la relación de las decisiones con los resultados obtenidos o simplemente para agregar más atractivo a nuestro producto.

Una historia es una hoja, de modo que se rige por los mismos métodos que se usan para crear hojas de trabajo y dashboards, ponerles nombres y administrarlos. Al mismo tiempo, una historia también es un conjunto de hojas dispuestas en un orden específico. Las hojas que componen una historia se denominan **puntos de la historia**.

La idea es que creemos una presentación con varias páginas las que van siguiendo una narrativa con introducción, desarrollo conclusión.

2. Interfaz de la pestaña de *historias*

En la siguiente imagen tenéis las principales partes de un historia:



historia

Aún así, tenéis [este](#) video donde tenéis más información sobre la interfaz de la pestaña de historias.

3. Crear una historia

En este caso, lo haremos prácticamente sobre nuestros datos. En [este link](#) tenéis el video que explica como montar nuestras historias.

Y hasta aquí las lecciones de Tableau .



Tableau-VIII-Historias.md 2KB

Binary

Descarga este fichero para tener la lección.

Proyecto 3

Este proyecto sigue el estilo de una competición Kaggle o un Datathon (Un hackaton relacionado con el mundo data). Os daremos unos datos que tenéis que limpiar, hacer un EDA, interpretar lo que tenéis, y hacer unas regresiones para predecir una variable en base de una o más variables. La empresa implementará el algoritmo que haya obtenido el resultado final del equipo con el RMSE más bajo, en sus servidores.

Descripción

La empresa GoGreen Bikesharing, se dedica al alquiler de bicicletas y posee datos tales como la cantidad de bicis alquiladas por usuarios registrados, la cantidad de alquileres realizados por usuarios puntuales, y la cantidad total. A estos datos se les añadió información meteorológica, y el calendario de festivos.

Ahora buscan analizar cuáles son los aspectos que más influyen en la cantidad de bicis que van a alquilar en un día.

Data Set



bikes.csv 62KB

Binary

El dataset para el proyecto del módulo 3.



Readme.txt 2KB

Binary

El documento explicativo del dataset.

Objetivos

En un o más Jupyter Notebooks, deberéis ir haciendo el análisis exploratorio de vuestros datos, la limpieza de los datos (eliminación de valores nulos, *outliers*, codificación de variables categóricas, normalización y estandarización, etc.) y todas aquellas técnicas que consideréis necesarias para obtener una mejor predicción.

Recordad dejar un jupyter limpio, sin resultados intermedios, y sin código que no sea necesario para el proyecto. Un desconocido debe entender el proceso que hayáis seguido y vuestros resultados.

Sprint 1

Haced un EDA, y formulad preguntas sobre estos datos que pueden merecer la pena investigar.

A mitad del proyecto tenéis que hacer una demo interna para enseñarle al *product owner* cómo habéis limpiado los datos, qué conclusiones preliminares salieron del EDA, y los hipótesis que os habéis planteado. No hace falta preparar un powerpoint.

Sprint 2

En el segundo sprint tenéis que enfocaros en el desarrollo del modelo en sí.

Entrega

Como es un trabajo en equipo no os olvidéis del marco *Scrum* y seguir la filosofía *Agile*. Usad un repositorio en GitHub en la organización Adalab para almacenar, desarrollar y entregar vuestro proyecto, siguiendo los nombres como en los proyectos anteriores, como por ejemplo:

- Adalab/project-da-promo-A-module-3-team-1
- Adalab/project-da-promo-D-module-3-team-3
- etc.

Fechas relevantes

- Inicio del proyecto y *sprint planning*: 18 Enero.
- Entrega del primer *sprint*, demo al *product owner*, y *sprint review*: 27 de Enero.
- Entrega del segundo *sprint*, y *sprint review*: 9 Febrero.
- Demo pública del proyecto: 13 de Febrero.

MODULO 4: Apoyo a la empleabilidad

Apoyo a la búsqueda de empleo

Descripción del módulo, fechas, metodología

Descripción del módulo

Una vez hayas acabado el curso intensivo, recibirás un módulo de cuatro jornadas llamado “Semana de la Empleabilidad” para apoyarte en la búsqueda de empleo.

Fechas

El detalle de las sesiones con su fecha, horario y enlace para acceder está disponible en tu calendario del curso en Calendar. Antes de empezar la Semana de la Empleabilidad, tendremos dos días llamados de “pre-work”, que utilizarás para ver sesiones en vídeo y realizar algunas actividades que te servirán para aprovechar las sesiones de la Semana de la Empleabilidad. De todas formas, cuando te gradúes se te facilitará en Slack y en tu Drive más información sobre este módulo.

Metodología

Durante la Semana de la Empleabilidad habrá tres tipos de sesiones:

1. Sesiones en vídeo
2. Sesiones en directo grupales
3. Sesiones en directo individuales

Detalles de las sesiones

Sesiones en vídeo

Antes de que empiece este módulo, te facilitaremos unos vídeos grabados por los facilitadores-as de la Semana de la Empleabilidad sobre desarrollo profesional.

Tendrás que ver los vídeos durante los días de pre-work.

Sesiones en directo grupales

Sesiones en vivo grupales

1. Introducción a la Semana de la Empleabilidad

Es una sesión impartida por el equipo de Student Success de Adalab, donde participarás con todas tus

companeras y en la cual se te explicara como funciona y se desarrolla la Semana de la Empleabilidad. Es una sesión muy útil, ya que podrás hacer preguntas y solventar cualquier duda sobre las sesiones y su dinámica.

2. Panel de antiguas Adalabers

Es un panel donde participarás con todas tus compañeras y donde conocerás la experiencia de algunas Adalabers de promociones anteriores que te hablarán sobre cómo organizaron su proceso de búsqueda de empleo, cómo fue su experiencia y cómo es su trabajo actual. Te darán consejos útiles para afrontar la búsqueda y aprovechar la Semana de la Empleabilidad.

3. Role play de entrevistas de Talento

Es una sesión impartida por la Career Coach de Adalab donde participarás con todas tus compañeras para practicar las entrevistas de Talento y recibir feedback. En esta sesión podrás poner en práctica los consejos que la Career Coach te ha brindado en sus vídeos.

4. Role play de entrevistas técnicas

Es una sesión donde participarás con tus compañeras en grupos reducidos para que practiques las entrevistas técnicas con algunos/as profesionales del sector IT que suelen realizarlas en sus empresas, para que entiendas cómo son y les pierdas el miedo.

5. Panel de Mentoring para las mentees

Es una sesión donde participarás con todas tus compañeras y que será impartida por la Responsable del Programa de Mentoring de Adalab, que te explicará en detalle cómo funciona el Mentoring y cómo aprovecharlo al máximo. Participarán también la Responsable de Selección de Mentores/as y algún antiguo mentor/a y su mentee para contar su propia experiencia con el programa.

6. Evento de Mentoring

Es el evento de lanzamiento del Programa de Mentoring de tu promoción, coordinado por la Responsable del Programa. Es el espacio donde podrás conocer a tu mentor/a en una sala privada de Zoom y aprovechar para intercambiar vuestros contactos.

7 . Retrospectiva de la Semana de la Empleabilidad

Es una sesión donde podrás darnos tu feedback sobre la Semana de la Empleabilidad. Tú y tus compañeras os reuniréis en grupos para decidir las 3 cosas a mejorar del qué (sesiones, materiales, etc.) y del cómo (cómo se han organizado e impartido las sesiones, etc.), y las 3 cosas a destacar del qué y del cómo con respecto a la Semana de la Empleabilidad, para luego trasladarlas al equipo de Student Success. Además, te pediremos que respondas a una encuesta sobre la Semana de la Empleabilidad.

Sesiones en directo individuales

En estas sesiones, trabajarás de forma individual con la Career Coach de Adalab que te facilitará consejos

y herramientas para potenciar tu marca personal de cara a la empleabilidad. Revisaras con ella tu CV, perfil de LinkedIn y podrás construir un buen relato profesional. Tendrás con ella dos sesiones: una primera de 20 minutos y una segunda de 15.

Es fundamental que llegues a estas sesiones con tu CV, relato profesional y perfil de LinkedIn ya trabajados para aprovechar al máximo el tiempo a disposición y los consejos que te ha facilitado la Career Coach previamente.

Sesiones en formato vídeo

Sesiones en vivo grupales

Sesiones en vivo individuales

Pre-work

Durante los días de pre-work tendrás que realizar algunas actividades:

- **Marca Personal**

Ve los siguientes videos

1. [Aprende a conocerte. Fortalezas de tu perfil profesional](#) (Lorena Ruiz, Career Coach)
2. [El relato profesional](#) (Lorena Ruiz, Career Coach)
3. [La búsqueda de empleo en el sector IT](#) (Lorena Ruiz, Career Coach)
4. [Consigue un CV potente y atractivo como Data Analyst](#) (Lorena Ruiz, Career Coach)

Siguiendo la información de los vídeos, **actualiza tu CV**. Esto te permitirá sacar más provecho a las sesiones individuales con la Career Coach.

Crear tu perfil en [EmMa](#) la plataforma donde más adelante accederás a las ofertas de empleo, siguiendo los pasos indicados en [esta guía](#). No te preocupes, que una vez hayas trabajado tu relato profesional en las sesiones de revisión individuales podrás actualizarlo en tu perfil.

- **Pon a punto tus redes**

Ve los siguientes videos

5. [Construye tu perfil profesional en LinkedIn como Data Analyst](#) (Lorena Ruiz, Career Coach)
6. [Potencia LinkedIn y aprovecha todas sus oportunidades](#) (Lorena Ruiz, Career Coach)
7. [Pon a punto tu GitHub para buscar empleo](#) (Ester Fernández, Data Analyst)
8. [Visibilízate como Data Analyst](#) (Laura Lacarra, Especialista en Comunidades)

- **El programa de Mentoring**

Ve el siguiente video:

9. [El programa de mentoring de Adalab](#) (Carmen Quesada, Responsable del Programa de Mentoring de Adalab)

Si tienes cualquier duda sobre los días de pre-work o la Semana de la Empleabilidad, ponte en contacto con el departamento de Student Success.

¡Ánimo con este nuevo comienzo!

MODULO 5: Post Bootcamp

¡Felicidades Analistas!

Si estás leyendo esto es porque has superado con éxito en el Curso Intensivo de Adalab...

¡¡¡tadaEnhorabuena Adalaber!!! Ahora toca ponerse a buscar trabajo, pero también es importante seguir estudiando y practicando programación con el objetivo de mejorar y de afianzar los conocimientos aprendidos durante el curso.

A partir de este momento ya no habrá clases y las profesoras no te podremos dar soporte pensive, porque estaremos ayudando a la siguiente promoción de Adalabers.

En este último módulo os proponemos información y recursos para cada una de vosotras, al ritmo que queráis, sigáis con vuestra formación.

Además en este repositorio (creado y mantenido por las alumnas de Adalab) disponéis de un montón de recursos muy útiles para que sigas aprendiendo por tu cuenta. También te animamos a que si encuentras un recurso, artículo... interesante, lo añadas al repositorio.

Movimiento de repos

Ya has acabado el curso intensivo de Adalab. Es hora de mover tus repos a tu usuaria de Adalab.

Durante el curso habrás creado unos cuantos repositorios en el GitHub de Adalab. Entre ellos:

- Repos de los proyectos de equipo
- Repos de las evaluaciones finales e intermedias
- Repos de prueba

Forkea los repos de tus proyectos de equipo

En Adalab nos interesa que los repos de tus proyectos se queden en la organización de Adalab porque los hemos publicado en redes sociales y si los movemos a otro sitio, la gente no podrá encontrarlos. Por ello te pedimos que los forkees.

¿Qué es hacer un fork?

En Git y GitHub **un fork es algo como hacer una rama de un repositorio pero en otro repositorio**. De esta forma nos queda el repositorio principal y el repositorio forkeado. Al ser dos repositorios independientes se puede trabajar de forma independiente, con programadoras independientes y **cada repo puede evolucionar por caminos separados**. Pero ambos repositorios siguen conectados por el fork, así que en un momento dado **podríamos mergear código desde el repo principal al repo forkeado**.

Los forks se utilizan mucho para hacer proyectos con variantes. Por ejemplo imaginemos que quiero crear mi propio navegador web pero no quiero programarlo desde cero porque eso sería muchísimo curro. Puedo coger el [repo de Chrome](#) (que es público pero yo no tengo permisos para modificarlo), hacer un fork en mi usuario de GitHub y empezar a cambiar el código desde ahí. Si dentro de dos meses Chrome añade nuevo código a su repo yo puedo mergear fácilmente ese código a mi repo forkeado y usarlo.

Si te [fijas en la esquina superior derecha del repo de Chrome](#) verás que la gente ha hecho varios miles de forks de este repo.

Forkea tus proyectos de equipo

Si en el futuro no vas a modificar los proyectos de equipo que has hecho durante el curso, no hagas nada.

Pero si quieres tener los proyectos de equipo en tu usuaria de GitHub para modificarlos o simplemente para tenerlos, haz lo siguiente:

- Entra en el repo de tu proyecto.
- Pulsa en la esquina superior derecha > **Fork**.
- Te pedirá que elijas en qué usuaria de GitHub quieres forkearlo. Elige tu usuaria.
- En ese momento tendrás en github.com/adalab/project-promo-x-module-x-team-x el repo principal y en github.com/mi-usuaria/project-promo-da-x-module-x-team-x el repo forkeado.
- Entra en github.com/mi-usuaria/project-promo-da-x-module-x-team-x, clónalo y haz lo que quieras, puedes cambiar el nombre al repo, cambiar el código, volver a desplegarlo en GitHub Pages...
- Por último y para que todo funcione bien, debes inhabilitar GitHub Pages y volverlo a habilitar para que GitHub genere correctamente la nueva URL.

Nota: no borres el repo original que está en el usuario Adalab.

Transfiere tus repos de las evaluaciones

El código de las evaluaciones es solo tuyo y lo lógico es que esté en tu usuaria de GitHub. Por ello lo vamos a mover.

Nota: La diferencia entre forkear y transferir es que transferir es simplemente mover un repo de una usuaria de GitHub a otra.

Busca los repos de tus evaluaciones y:

- Entra en las **Settings**. Si no tienes acceso a las settings de tu repo pídele ayuda a una de las profes.
- En la parte inferior **Danger Zone** verás el botón **Transfer ownership**. Púlsalo.
- Te pedirá que escribas el nombre de la usuaria a la que quieras transferirlo. Escribe tu nombre de usuaria.
- Es posible que te envíen un correo para confirmar la operación.
- A partir de ese momento el repo desaparecerá del GitHub de Adalab y aparecerá en tu GitHub.

Borra los repos que no quieras

El resto de repos que has creado durante el curso, por ejemplo para hacer prácticas de merges y ramas, hay que borrarlos o transferirlos.

Si los quieres conservar, transfírelos a tu usuaria de GitHub. Si no te interesan para nada bórralos desde las **Settings > Danger zone > Delete this repository**.

Borrado automático de repos

Con el objetivo de mantener limpio el GitHub de Adalab, ya sabes que somos gente limpia y ordenada, hemos creado un bot que transcurrido un mes desde que acaba una promoción borra todos los repos de las alumnas, excepto los repos de los proyectos de equipo.

Si transcurrido ese mes no has transferido los repos a tu usuaria de GitHub los perderás para siempre jamás .

Ruta Data Engineer y Data Analyst

¿Qué hace una Data Engineer?

Una Data Engineer o Ingeniera de datos se especializa en el procesado de grandes volúmenes de datos en crudo desde las fuentes como bases de datos, API's o archivos de texto. Y se encargan de automatizar estos procesos desarrollando lo que se llaman como ETL (Extracción, Transformación y carga de datos) para que otros roles del campo de datos como pueden ser las Data Scientist o Científicas de datos, así como las Machine Learning Engineers o Ingenieras de Aprendizaje Máquina puedan utilizar estos datos para resolver otra clase de problemas.

Algunas de las herramientas tecnológicas utilizadas por las Ingenieras de datos puede incluir:

- Saber algún lenguaje de programación, como en nuestro caso Python.
- Uso de Bases de Datos SQL o noSQL
- Herramientas de procesado y flujo de datos como Apache Spark, Airflow o servicios cloud como AWS o Azure

1. Big Data y Cloud Computing

- [Curso de Introducción a AWS: Fundamentos de Cloud Computing](#)
- [Curso de Introducción a AWS: Cómputo, Almacenamiento y Bases de Datos](#)
- [Curso de Introducción a AWS: Redes, Gobernanza y Machine Learning](#)
- [Curso Práctico de AWS: Roles y Seguridad con IAM](#)
- [Curso Práctico de AWS: Cómputo con EC2](#)
- [Curso Práctico de Storage en AWS](#)
- [Curso Práctico de Bases de Datos en AWS](#)
- [Curso de Big Data en AWS](#)
- [Curso de AWS Redshift para Manejo de Big Data](#)

2. Business Intelligence

- [Power BI](#)

Ruta Data Scientist

¿Qué hace una Data Scientist?

Una Data Scientist es una persona que conoce el negocio y utiliza los datos para responder preguntas que aporten valor a la organización. Desarrolla modelos de Aprendizaje máquina con el fin de ser capaz de resolver algunos problemas como la predicción de nuevos usuarios, sistemas de recomendación, creación de perfiles de usuario, entre muchas otras aplicaciones.

Algunas de las herramientas tecnológicas utilizadas por las Ingenieras de datos puede incluir:

- Saber algún lenguaje de programación, como en nuestro caso Python.
- Uso de Bases de Datos SQL o noSQL

Adicionalmente tiene unos conocimientos de matemáticas más extensos que por ejemplo una Data Analyst o una Data Engineer, ya que todo el aprendizaje máquina tiene por debajo una base matemática muy fuerte para dar sentido a los algoritmos aplicados sobre los conjuntos de datos.

1. Fundamentos estadísticos y matemáticos

- [Curso de Funciones Matemáticas para Data Science e Inteligencia Artificial](#)
- [Curso de Matemáticas para Data Science: Estadística Descriptiva](#)
- [Curso de Matemáticas para Data Science: Probabilidad](#)
- [Curso Básico de Cálculo Diferencial para Data Science e Inteligencia Artificial](#)
- [Curso de Estadística Inferencial para Data Science e Inteligencia Artificial](#)
- [Curso de Fundamentos de Álgebra Lineal con Python](#)
- [Curso de Álgebra Lineal Aplicada para Machine Learning](#)

2. Machine Learning con Python

- [Curso de Fundamentos Prácticos de Machine Learning](#)
- [Curso Profesional de Machine Learning con Scikit-Learn](#)

3. Deep Learning

- [Curso de Fundamentos de Redes Neuronales con Python y Keras](#)
- [Curso de Redes Neuronales Convolucionales con Python y Keras](#)
- [Curso Profesional de Redes Neuronales con TensorFlow](#)
- [Curso de Deep Learning con Pytorch](#)
- [Curso de Transfer Learning con Hugging Face](#)
- [Curso de Introducción a Demos de Machine Learning con Hugging Face](#)

- [Curso Profesional de Computer Vision con TensorFlow](#)
- [Curso de Fundamentos de Procesamiento de Lenguaje Natural con Python y NLTK](#)
- [Curso de Algoritmos de Clasificación de Texto](#)

Desarrollo Profesional

Estos cursos son una ruta para ampliar tus conocimientos en desarrollo profesional. No es necesario que los realices todos, lo que te ofrecemos es una guía para seguir estudiando al acabar Adalab.

1. Cursos para aumentar la productividad, gestionar las emociones y mejorar la comunicación:
 - [Curso de Gestión Efectiva del Tiempo](#)
 - [Curso para Crear Hábitos Positivos](#)
 - [Curso de Estrategias para la Productividad y la Organización](#)
 - [Curso de Herramientas para el Crecimiento Personal](#)
 - [Audiotaller de Resolución de Problemas](#)
 - [Curso de Manejo de Emociones para la Productividad y la Organización](#)
 - [AudioCurso de Comunicación Efectiva](#)
 - [Curso de Comunicación Asertiva](#)
 - [Curso de Networking Efectivo](#)
 - [Curso de Oratoria para Hablar en Público](#)
 - [Curso para Superar el Síndrome del Impostor](#)
 - [Curso para Desarrollar la Autoconfianza](#)
 - [Curso de Inteligencia Emocional](#)
 - [Curso Herramientas Prácticas para la Inteligencia Emocional](#)
 - [Curso de Manejo de Ansiedad con Terapify](#)
 - [Curso de Manejo de la Frustración](#)
 - [Curso para Desarrollar la Empatía](#)
 - [AudioCurso de Cómo Recibir Feedback](#)
2. Cursos para enfrentarse a la búsqueda de empleo:
 - [Curso de Creación de Portafolio y CV](#)
 - [Curso de Optimización del Perfil de LinkedIn](#)
 - [Curso para Triunfar en Entrevistas de Trabajo](#)
 - [Curso para Conseguir Trabajo en Tecnología](#)
 - [Curso de Inglés para Entrevistas de Trabajo](#)
 - [Curso de inglés para developers](#)
3. Cursos para trabajar en equipo:
 - [Curso de Motivación para Equipos de Trabajo](#)
 - [Curso de Desarrollo de Habilidades Blandas para Equipos de Trabajo](#)
 - [Curso de Liderazgo para Equipos de Trabajo](#)
 - [Curso de Comunicación Efectiva para Equipos de Trabajo](#)