**Sesión 2 ---**

**LINUX**

Árbol jerárquico por el que te vas moviendo:

* Nodo raíz **/**
* **Bin**  carpeta donde tenemos ejecutables (comandos)
  + **Cd ..**  carpeta a la que queremos subir
  + **Cd (nombre carpeta queremos ir)**  carpeta a la que queremos bajar
    - Rutas relativas
  + **Cd /(carpeta a la que queremos ir)**  salto de carpeta, estés donde estés
    - Ruta absoluta
  + **Ls (nombre carpeta)**  listar carpetas dentro
  + **Ls -l (nombre carpeta)**  si es carpeta hay un d (de directorio) al principio.
  + **Whereami**  ¿?
  + **Pwd**  para saber dónde estoy
  + EN VERDE FICHERO, EN AZUL CARPETA
    - Por debajo de un fichero no hay nodos, son nodos raíz.
* **Log**  ficheros donde se escribe todo lo que ha hecho una app
* **Lib**  conjunto de funciones para hacer algo de un tercero
* **Home**  carpeta donde estás cuando arrancas
* **Root**  carpeta de nuestro user
  + Py
  + C
  + Js
  + Txt
* **Opt** 
  + Temu  directorio porque hay una **d** al principio

Para crear fichero: **touch nombre fichero + extensión**

Para borrar fichero: **rm nombre fichero + extensión** (rm de remove)

* No hay comando para deshacer, OJO

Para crear carpeta: **mkdir + nombre carpeta**

Para moverse entre comando utilizados  flechas

Para mostrar opciones o autorellenar  TAB

Para borrar carpetas y ficheros  **rm -rf nombre carpeta**

Cuando no sabes un comando lo que es: **lo escribes** **--help** y te dice lo que es y sus extensiones

**Cp**  copiar

* Si estamos arriba: Cp ruta + fichero ESPACIO ruta destino + fichero
* Si estamos bajo: **Cp fichero ESPACIO ../ruta destino + fichero** (subes un nivel y bajas)

Mover  **mv**

**Echo** “xxxxxxxxx” escribir en pantalla

**Echo “xxxxxxx” > nombre fichero**  escribir en el fichero

**Cat nombre fichero**  muestra el contenido /// **nano / vim¿?** 🡪 ver y editar contenido fichero

**Tail nombre fichero**  muestra ultimas diez líneas

**Ctrl + C**  cancelar comando

Cambiar permisos  **chmod 777** (todos los permisos) **+ nombre fichero**

Cambiar permisos de un grupo a otro  **chgrp**

**Sudo + comando cualquiera 🡪** ejecutarlo como admin

GOOGLE CLOUD

Alternativas: AWS (Amazon) y Azure (Microsoft)

IAAS  infraestructura como servicio (ordenador en la nube)  
PAAS  plataforma como servicio  
SAAS  software como servicio (software de terceros que se instala en la nube, ej. Gmail)

**BASH**

IF / FOR / WHILE ¿?????????¿¿¿¿¿¿¿¿¿¿

**Sesión 3 ---**

**GITHUB**

**Solución en la nube gratuita, permite crear repo de códigos públicos y privados**

**Basado en GIT 🡪 control versiones**

**Alternativas: gitlab, bitbucket (pago)**

**Trabajar con GitKraken:**

* **Create Branch**
* **Hacer commit de los cambios**

**Hacer Pull Request (PR) para integrar cambios en main (en github.com)**

**Sesión 4 ---**

**Máquinas virtuales**

**CI/CD**

**Vagrant**

**Microservicios** 🡪 uso de contenedores, dividir en distintos “servicios” a la aplicación 🡪 útil para su desarrollo y posterior uso, al poner en servidores distintos según la capacidad que requieran(mirar PPT teoría)

**Contenedores vs Máquinas virtuales** (mirar PPT teoría)

**Sesión 5 ---**

**DOCKER** (mirar PPT teoría)

Microservicios: problema de la integración entre servicios  transaccionalidad

**$ docker run -dp 80:80 docker/getting-started**

* Docker run  crea contenedor
* -d  ejecutar en segundo plano
* P 80:80  puerto, mapear puerto 80 del contenedor al puerto 80 de la maquina anfitriona.

**$ docker ps -a**  listar contenedores

A raíz de **Dockerfile** se crea imagen (archivo sin extensión)

**$ docker build -t second\_edem\_img .**  crear imagen

**$ docker exec -it cont\_second\_edem bash**  ejecutar + contenedor + comando bash  abrir/entrar en el contenedor

**$ cd /var/www/html**  entrar en el archivo

**$ echo “<p>my\_name</p>” >> ./index.html**  escribir en el archivo ./index.html

**$ cat index.html**  ver contenido de index.html

**$ docker images**  lista imágenes

**Echo $USER**  devuelve usuario con el que estás

**Exit** para salir de máquina

**$ docker rm + nombre contenedor**  borrar contenedor

**$ docker rmi + nombre imagen**  borrar imagen

**$ docker run -itd --name cont\_third\_edem -p 8080:80 third\_edem\_img**

<http://localhost:8080>

**$ docker commit cont\_exercise\_7 cont\_fourth\_edem\_img\_newline:latest ** a partir del contenedor creas una imagen, tras la modificación realizada

**$ docker login --username=[Dockerid]**  hacer login en DockerHub con usuario

Crear en DockerHub repositorio  lo llamamos “myfirstimage”

**$ docker tag [image\_id] [Dockerid]/myfirstimage:latest ** asocias a la imagen un tag, en este caso “latest” (última imagen)

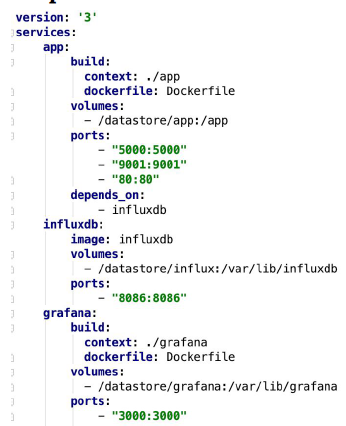
**$ docker push [Dockerid]/myfirstimage ** haces push al repositorio

**$ docker cp + nombre archivo 🡪** copia un fichero entre tu ordenador y el contenedor (o al revés)

Docker volume

Almacenar datos de contenedores en máquina anfitriona (mirar PPT teoría)

Docker compose



Archivos con extensión **.yml** (yaml)

Importancia de jerarquías y tabulaciones

Ej. Servicios:

* App
  + Construir a partir de dockerfile
  + Crea volumen
  + -p puedes poner varias redirecciones
  + Crear dependencias
* Influxdb
  + No construye imagen, directamente va a por ella.
  + Primero busca en local y luego va a DockerHub
* Grafana = app

Nos pasarán este fichero y levantaremos la estructura necesaria para trabajar

Habrá que hacer un $ y trabajar

**$ docker compose up -d ** en la ruta donde esté el archivo

**Kubernetes**

Desplegar gran cantidad de contenedores (mirar PPT teoría)

**PROGRAMACION (Sesión 1.html)**

StackOverflow 🡪 foro de programación

Replit.com 🡪 entorno online de programación, varios lenguajes disponibles, es público

Main.py 🡪 archivo principal que empieza a ejecutar las cosas

Errores:

* En tiempo de compilación: código básico del script con error, en funciones nativas
* En tiempo de ejecución: código de las variables/fórmulas planteadas

**Print(value, opcionales varios)** 🡪 función nativa (builtin), muestra por consola lo que queramos

En Python y en JS da igual utilizar comillas simples **‘** o dobles **“** 🡪 indican texto, da igual, lo único es que debe coincidir inicio y cierre

**=** 🡪 asignación de un contenido a una variable, contenido que se almacena en una dirección de memoria concreta

* En los datos primitivos, se pasa el valor por copia 🡪 fuera no se replica
* En los datos complejos, se pasa por referencia

Para concatenar en print 🡪 **+**

Para concatenar en print v2 🡪 **f** y luego incluir la variable entre **{}** (usar en tipos primitivos)

Si especificas el tipo de dato, se queda guardado y no se puede modificar, si no es tipado dinámico, puede cambiar más tarde.

**‘’’** comentario **‘’’**🡪 tres comillas simples, comentarios multilínea 🡪 poner para apertura y cierre

**#** 🡪 comentario de una línea

* **# TODO :** Tarea pendiente de hacerse
* **# FIXME :** Correcciones de código necesarias

miCadena: str = "Hola Mundo"

miCadena2: str = '''Hola Mundo'''

Obtener “Ho”

primeros\_dos\_caracteres = miCadena[0:2] 🡪 el corchete final es abierto, no incluye la posición final

print(primeros\_dos\_caracteres)

Colocar en mayúsculas

print(f'Texto Con mayusculas: {miCadena.upper()}')

La primera en mayúsculas

print(f'Texto Con mayusculas: {miCadena.capitalize()}')

Todo en minúsculas

print(f'Texto Con mayusculas: {miCadena.upper()}')

miLista: [str] = ['Martin','Juan','Ana']

print(miLista)

print(\*miLista) 🡪 imprime la lista sin corchetes ni comas

miRango = range(1, 10, 2) 🡪 números pares

print(miRango) 🡪 NO funciona 🡪 utilizar \_\_str\_\_ o \_\_repr\_\_

utilizar print(\*miRango)

Diccionarios:

persona = {

"nombre": "Julen",

"edad": 26,

"email": "julen"

}

print(persona) 🡪 NO funciona

print(\*persona) 🡪 representa las claves

print(f'La edad de {persona["nombre"]} es {persona.get("edad")}')

--iterable-- 🡪 función que va devolviendo valores de una lista

* En una función “set” te pide un valor iterable, que esté en una iteración de datos o lista.
* [\_T] 🡪 es un genérico, cualquier tipo de dato.

misNumeros = [1,2,3,3,4,5,1,5]

print(\*misNumeros)

miSetDeNumeros = set([1,2,3,3,4,5,1,5]) 🡪 hace que no se repitan los datos, desordenado

print(miSetDeNumeros)

miOtroSetDeDatos = {1,2,2,5,5,'a',False}

print(miOtroSetDeDatos) 🡪 los ordena

frozenset() 🡪 set inmutable, no varía

Palabras reservadas 🡪 tipos de datos:

* Str
* Int
* Float
* Complex 🡪 números complejos
* List
* Tuple
* Range
* Dict
* Set
* Frozenset
* Bool
* Bytes 🡪 valores binarios
* Bytearray
* Memoryview
* Global + variable 🡪 variable que será accesible en todo el espectro, dentro de funciones y fuera de ellas 🡪 cualquier referencia a esa variable será a la variable inicialmente determinada

miValorNone = None

print(miValorNone)

resultado\_operacion: float = (((6-2)/5)\*\*2) 🡪 \*\* elevado a

print(round(resultado\_operacion,0)) 🡪 round redondea

print(resultado\_operacion.\_\_round\_\_(0)) 🡪 exactamente lo mismo que anterior, esta función es del DataModel de Phyton

RETO GRUPAL 1

Partiendo de:

- cantidad a invertir

- el interés anual

- El número de años

Calcular el capital obtenido con dos decimales.

invertir = 5000

interes = 0.02

años = 2

capital = ((invertir\*interes)\*años)+invertir

print("El capital obtenido es "+ str(capital) + "€")

RETO GRUPAL 2

producto = 14.99

Los que no son de temporada tendrán 30% descuento

Programa que reciba un número de productos en temporada, otro fuera de y aplicar el descuento para obtener el gasto total.

Indica la cantidad ahorrada

precio=14.99

descuento=0.3

temporada = input('¿Cuántos productos de temporada tienes? ')

temporada = int(temporada)

fuera = input('¿Cuántos productos fuera de temporada tienes? ')

fuera = int(fuera)

gasto = (temporada\*precio)+(fuera\*precio\*(1-descuento))

ahorro = round(((temporada+fuera)\*precio)-gasto,2)

print('La cantidad ahorrada gracias al descuento es ' + str(ahorro) + ' €')

RETO GRUPAL 3 -- AÑO BISIESTO

1. Introduce un año.

2. Que diga si es bisiesto o no

3. Que pida años hasta que el usuario introduzca un año bisiesto.

4. Mensaje de exito

año = int(input('Introduce un año '))

if año % 4 == 0 and (año % 100 != 0 or año % 400 == 0):

  print("Es bisiesto")

else:

  print("No es bisiesto, prueba otra vez")

* == 🡪 es igual a
* != 🡪 no es igual a
* and = & 🡪 se pueden usar los dos

'''

RETO GRUPAL 3 -- AÑO BISIESTO

1. Introduce un año.

2. Que diga si es bisiesto o no

3. Que pida años hasta que el usuario introduzca un año bisiesto.

4. Mensaje de exito

'''

while True:

año = int(input('Introduce un año '))

if año % 4 == 0 and (año % 100 != 0 or año % 400 == 0):

   print("Es bisiesto")

break

else:

  print("No es bisiesto, prueba otra vez")

'''

RETO GRUPAL 4

Consola pide al usuario los datos de una peli

- Nombre (str)

- Año (int)

- Puntuación (float, debe ser entre 1 y 5 estrellas)

- Genero (str, solo puede ser comedio, terror, acción y drama, crear lista y comprobar que esté dentro)

Guardar todo en diccionario

'''