

Práctica 0: Introducción

Algoritmia y optimización

Curso 2025–26

Índice

1. Introducción	2
2. Docker	2
2.1. Primeros pasos	3
2.2. Instalación	3
2.3. Docker Images	4
2.4. Docker Containers	6
2.5. Dockerfile	6
2.6. Docker Volumes	7
2.7. Entorno	8
3. Introducción a Python	12
3.1. Variables	12
3.2. Tipos de datos	12
3.3. Cadenas	13
3.4. Operadores	13
3.5. Estructuras de control	13
3.6. Estructuras de repetición	14
4. Introducción a Matplotlib	14
4.1. Crear un gráfico básico	14
4.2. Personalizar el gráfico	15
4.3. Crear múltiples gráficos	15
4.4. Gráficos 3D	15

1. Introducción

En esta práctica 0 haremos un repaso de las tecnologías y conocimientos necesarios para un mejor desarrollo de las prácticas de la asignatura “Algoritmia y optimización”.

2. Docker

En los primeros días de la computación, era común utilizar un servidor dedicado para cada aplicación. Esto se debía a la falta de tecnologías suficientemente seguras y eficientes que permitieran la ejecución de múltiples aplicaciones en un solo servidor. Esta limitación obligaba a las empresas a adquirir nuevos servidores para cada aplicación que se desarrollaba, lo que resultaba en un uso ineficiente de los recursos, con servidores operando al del 5 % al 10 % de su capacidad total.

Con la aparición de VMware y las máquinas virtuales (MMVV), se revolucionó el uso de servidores. Las MMVV permitieron ejecutar múltiples aplicaciones de manera segura en un único servidor físico. Cada máquina virtual actúa como un sistema independiente, ejecutando su propio sistema operativo (SO), lo que permite una mayor densidad de aplicaciones en un solo hardware.

Sin embargo, cada MV requiere un sistema operativo completo, lo que implica un consumo significativo de recursos como CPU, RAM y almacenamiento. Además, en muchos casos, los sistemas operativos necesitan licencias, lo que añade costos adicionales y complejidad administrativa. Otro inconveniente de las máquinas virtuales es su lenta velocidad de arranque y la limitada portabilidad, lo que dificulta moverlas entre diferentes plataformas, incluyendo entornos en la nube.

La evolución hacia el uso de contenedores ha ofrecido soluciones significativas a las limitaciones de las máquinas virtuales. Los contenedores son entornos de ejecución ligeros que empaquetan una aplicación junto con sus dependencias, pero comparten el kernel del sistema operativo del host. Esto reduce drásticamente la sobrecarga de recursos, ya que no requieren un sistema operativo completo para cada contenedor.

Los contenedores son extremadamente eficientes, permitiendo un arranque casi instantáneo y una alta portabilidad, facilitando su traslado entre sistemas locales y plataformas en la nube. Esta independencia del sistema operativo maximiza el uso de los recursos del servidor y simplifica el desarrollo y la implementación de aplicaciones.

Docker ha sido fundamental en la popularización y democratización del uso de contenedores. Desde su introducción, Docker ha facilitado la creación, gestión y orquestación de contenedores, haciendo que esta tecnología sea accesible tanto para grandes empresas como para pequeños desarrolladores. Muchas empresas han contribuido al desarrollo de la tecnología de contenedores, ampliando sus capacidades y aplicaciones en diversos entornos de TI.

2.1. Primeros pasos

Docker es un programa que crea, organiza y orquesta contenedores. Está construido a partir de diversas herramientas con origen en el *Moby project* de código abierto. Docker, Inc. es la compañía que creó Docker y continúa manteniendo la solución que hace de los contenedores una herramienta fácil de emplear.

2.2. Instalación

Visita la web <https://www.docker.com/> e instala Docker Desktop. Una vez instalado, ábrelo. Ya tendremos acceso por terminal a los comandos asociados a Docker. Ejecuta el siguiente comando para comprobar que se ha instalado correctamente.

```
docker version
```

💡 Consejo

En linux puede ser que tengas que añadir tu usuario al grupo `docker` con:
`usermod -aG docker <user>`

La salida debería verse parecida a esto:

```
Client:
 Version:           27.0.3
 API version:       1.46
 Go version:        go1.21.11
 Git commit:        7d4bcd8
 Built:             Fri Jun 28 23:59:41 2024
 OS/Arch:           darwin/arm64
 Context:           desktop-linux

Server: Docker Desktop 4.32.0 (157355)
Engine:
 Version:           27.0.3
 API version:       1.46 (minimum version 1.24)
 Go version:        go1.21.11
 Git commit:        662f78c
 Built:             Sat Jun 29 00:02:44 2024
 OS/Arch:           linux/arm64
 Experimental:      false
 containerd:
```

```
Version:      1.7.18
GitCommit:    ae71819c4f5e67bb4d5ae76a6b735f29cc25774e
runc:
Version:      1.7.18
GitCommit:    v1.1.13-0-g58aa920
docker-init:
Version:      0.19.0
GitCommit:    de40ad0
```

2.3. Docker Images

Las imágenes de Docker son objetos que contienen un Sistema Operativo (SO), una aplicación y sus dependencias. Es como una plantilla de una aplicación. Ejecuta el siguiente comando:

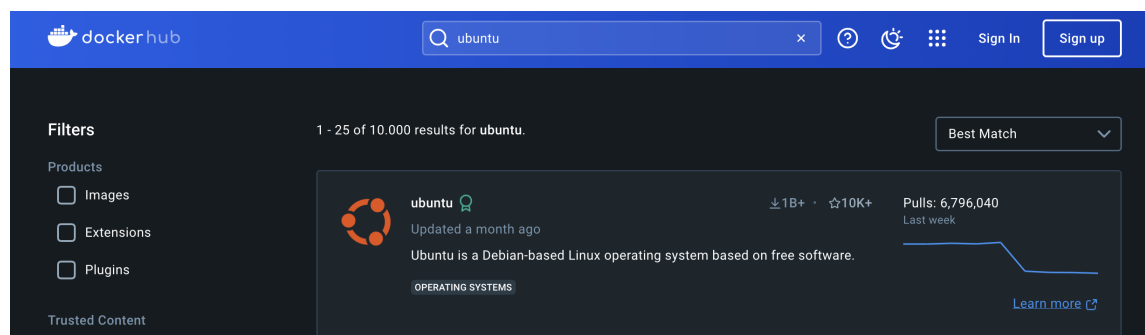
```
docker images
```

La salida debe ser:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

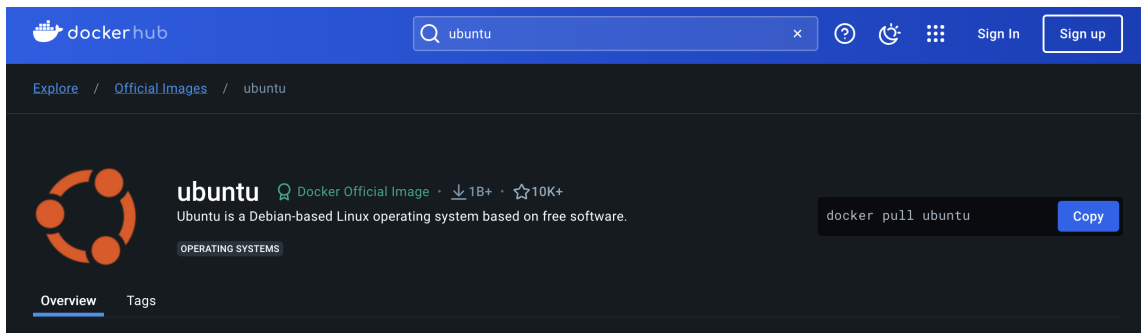
Para descargar alguna imagen primero debemos visitar la web Docker Hub donde encontraremos un amplio catálogo de imágenes de Docker prediseñadas listas para emplear.

Si buscamos Ubuntu, encontraremos la **imagen oficial**¹ del sistema operativo (con más de 1 billón de descargas).



Si accedemos a los detalles, en la cabecera encontraremos el comando necesario para descargar la imagen oficial en nuestra máquina.

¹Es importante verificar que imagen que emplearemos es oficial, ya que nos garantiza una serie de estándares de seguridad.



```
docker pull ubuntu
```

Sin embargo, es importante conocer que Docker emplea la nomenclatura:

nombre:tag

Mediante este sistema podemos obtener una versión específica de la imagen. Como por ejemplo:

```
docker pull ubuntu:24.04
```

Veremos el proceso de descarga. Para cerciorarnos de que se ha descargado correctamente la imagen, podemos visualizar las imágenes de nuestro sistema con:

```
docker image ls

# alternativa, hace lo mismo
docker images
```

Veremos:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	oracular-20240617	098a556a9a5d	6 weeks ago	101MB

Para eliminar una imagen de nuestro ordenador emplearemos:

```
docker rmi <ID>
```

Hasta ahora hemos visto qué es una Imagen de Docker (una plantilla). Por lo tanto, un contenedor será una instancia o un entorno a partir de una imagen. Tendremos una única

imagen de `ubuntu:oracular-20240617` en nuestro ordenador pero un número infinito de contenedores empleando esa imagen.

2.4. Docker Containers

Para lanzar un contenedor a partir de una imagen empleamos el comando `run`:

```
docker run -it ubuntu:oracular-20240617 /bin/bash
```

Consejo

Puedes añadir, y es muy recomendable, el flag `--name` para asignar un nombre personalizado al contenedor

Los flags `-it` indican que el contenedor debe ser interactivo y tiene que incrustarse en la sesión de terminal que estamos. Para salir del contenedor sin pararlo presiona `Ctrl+P+Q`. Para volver a entrar en el contenedor debemos emplear el comando `docker exec` :

```
docker exec -it <ID> bash
```

El `<ID>` de nuestro contenedor no es necesario que sea completo, podemos emplear los `N` primeros caracteres que hagan del contenedor único. Si el ID fuera `3a43ce646149`:

```
docker exec -it 3a bash
```

Para parar y eliminar un contenedor, sal de él con `Ctrl+P+Q` y escribe:

```
docker stop <ID>
docker rm <ID>
```

Advertencia

Comprueba con `docker ps -a` si el contenedor ha sido eliminado correctamente.

2.5. Dockerfile

El *Dockerfile* es un fichero de texto plano que le dice a Docker cómo construir una aplicación y qué dependencias incluir en la imagen.

Un *Dockerfile* básico para trabajar con Python podría tener la siguiente estructura:

```
# Imagen base
FROM python:bullseye

LABEL maintainer="jcmartinez.sevilla@ua.es"

# Cambiamos el directorio de trabajo
# (pongo /workspace porque los volúmenes los
# suelo montar aquí, en el siguiente apartado más info)
WORKDIR /workspace

# Instalamos la última versión de pip
RUN pip install --upgrade pip

# Copiamos el fichero requirements.txt dentro del contenedor
COPY requirements.txt ./

# Instalamos dependencias
RUN pip install --no-cache-dir -r requirements.txt
```

Información

En nuestro `requirements.txt` hemos incluido el paquete `matplotlib==3.9.1`

Para construir una imagen a partir de un *Dockerfile*:

```
docker build -t mi_imagen:1.0 .
```

Con `-t` indicamos una etiqueta o tag para nuestra imagen y con `.` indicamos que el *Dockerfile* se ubica en el propio directorio donde estamos lanzando el comando.

Anteriormente hemos hablado de las Docker Images, las cuales hacen la función de plantilla. Es necesario añadir que están formadas por capas. Esto permite que en el caso de sufrir alguna modificación, sólo se añadirán o se eliminarán las capas necesarias.

2.6. Docker Volumes

Existen dos tipos de datos, persistentes y no persistentes. Para tener datos persistentes en Docker necesitamos *Volumes* (volúmenes en castellano). Los volúmenes son independientes del ciclo de vida de los contenedores. Además, pueden ser compartidos por varios contenedores. Para simplificar este aspecto de Docker, añadiremos a nuestro comando `docker run` lo siguiente:

```
docker run -it --name <nombre_custom_contenedor> -v
<path_absoluto_a_carpeta>:<carpeta_dentro_contenedor>
<nombre_imagen_docker> /bin/bash
```

Un ejemplo podría ser:

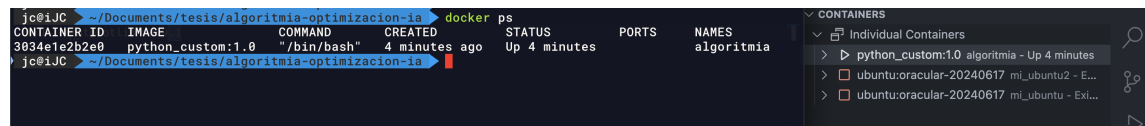
```
docker run -it --name mi_imagen -v
/home/jcms/mydata/./workspace mi_imagen:1.0 /bin/bash
```

En este punto deberías saber qué es un imagen de Docker, un contenedor, un Dockerfile, los volúmenes y cómo trabajar con ellos.

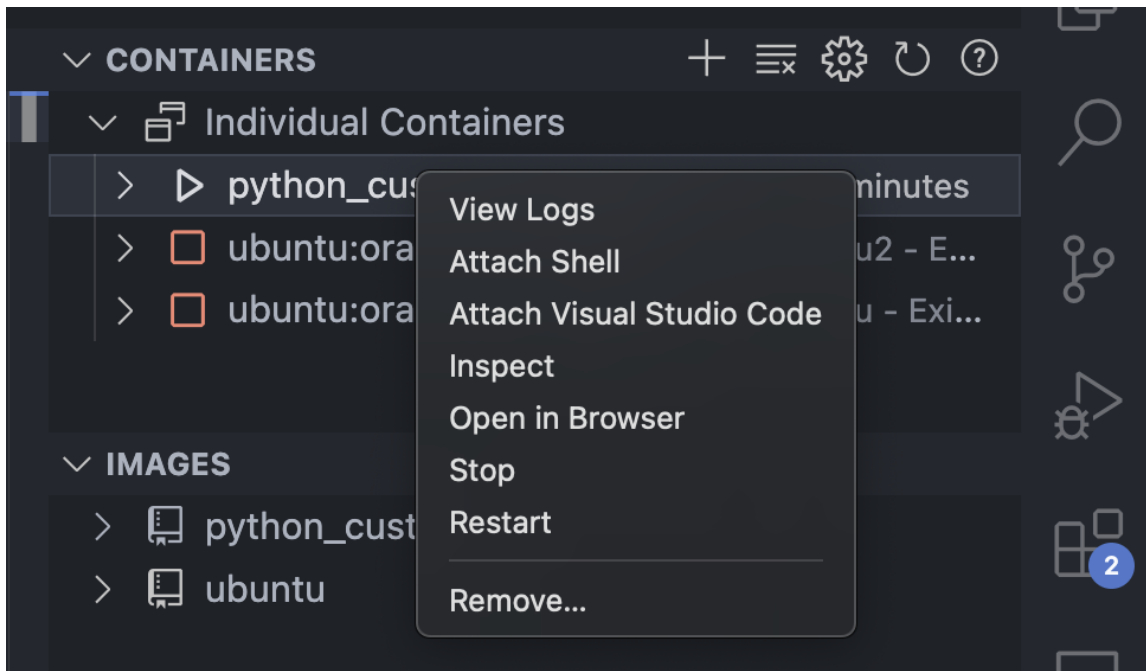
2.7. Entorno

Una vez creado nuestro *Dockerfile* y lanzado nuestro contenedor junto con el volumen asociado, podemos ir a nuestro IDE de confianza. En mi caso, trabajo con Visual Studio Code, donde instalaremos la extensión oficial de Docker.

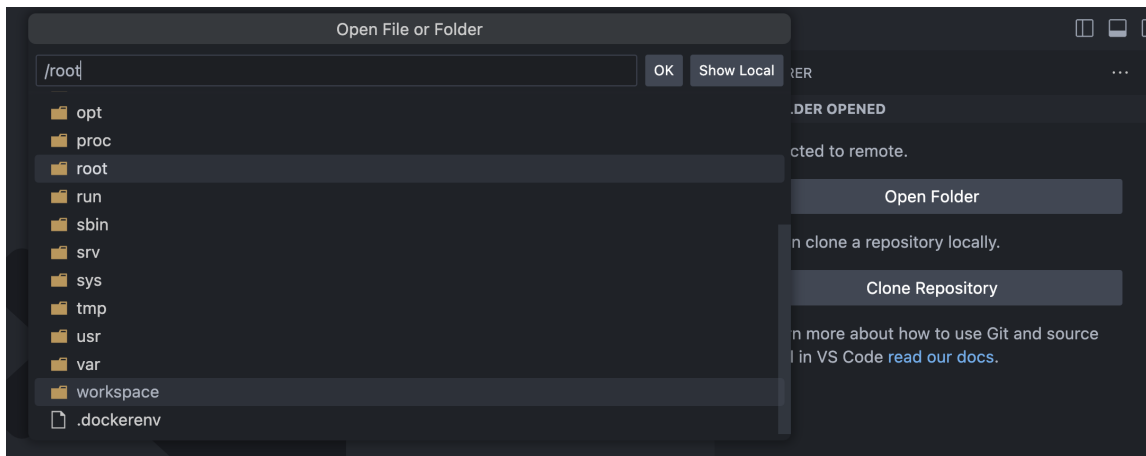
Una vez instalada, accedemos desde la barra lateral al icono de Docker, donde aparecerán los contenedores activos.



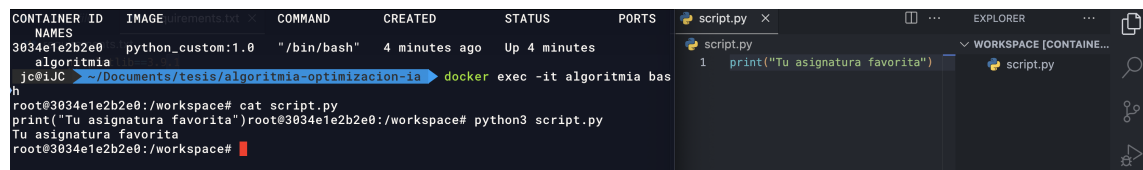
Si hacemos click derecho en VSCode en el nombre de nuestro contenedor, aparecerá la opción *Attach Visual Studio Code*, hacemos click en ella:



En la ventana emergente, seleccionamos **Open Folder** y la carpeta en la que tendremos nuestros archivos, en este ejemplo **workspace**:



Una vez dentro de nuestra carpeta, como hemos descargado una imagen de Docker con Python incluido, podemos ejecutar scripts.

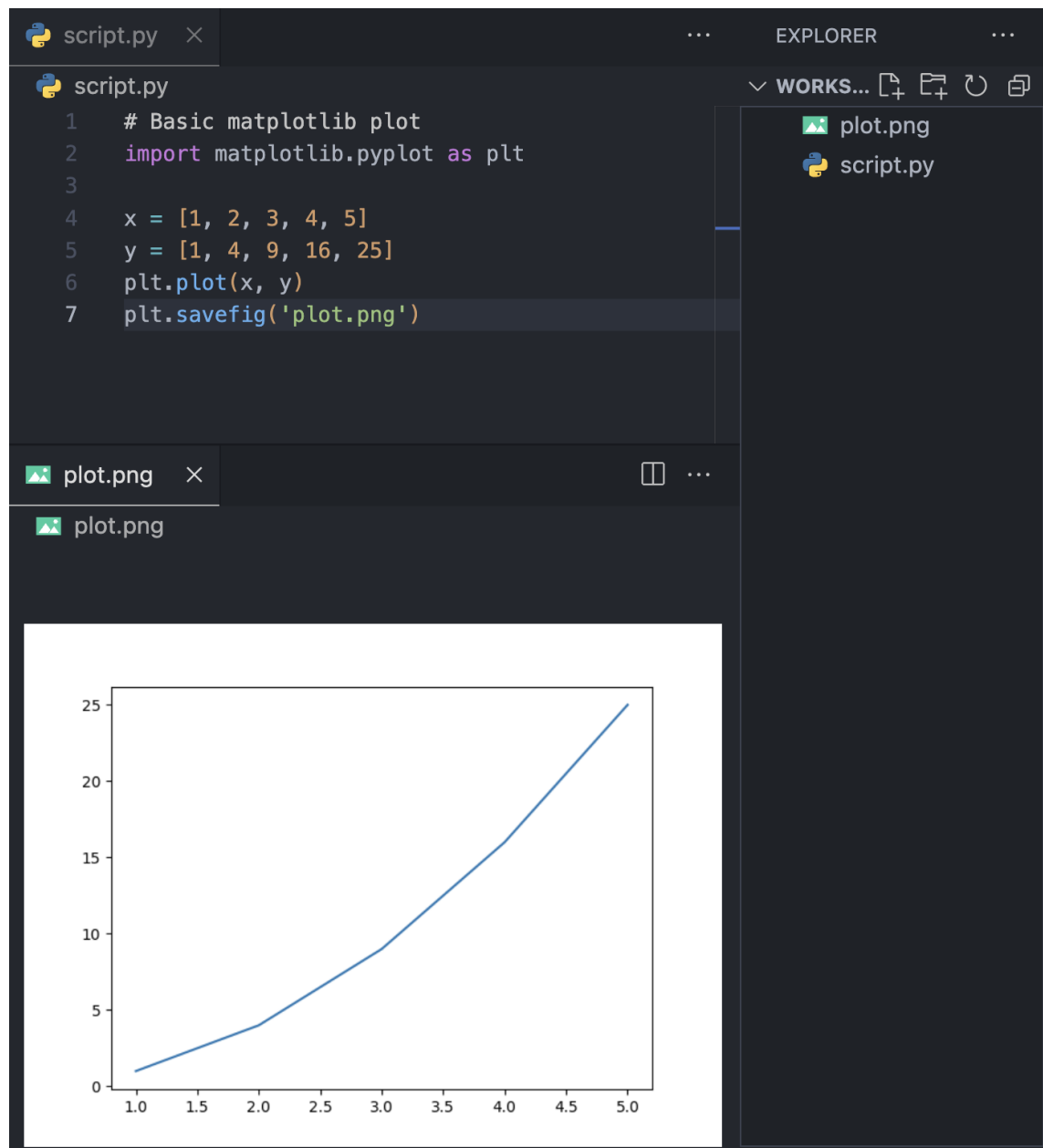


```
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
3034e1e2b2e0   python_custom:1.0   "/bin/bash"             4 minutes ago   Up 4 minutes
jce1JC         ~/Documents/tesis/algoritmia-optimizacion-ia   docker exec -it algoritmia bash
root@3034e1e2b2e0:/workspace# cat script.py
print("Tu asignatura favorita")root@3034e1e2b2e0:/workspace# python3 script.py
Tu asignatura favorita
root@3034e1e2b2e0:/workspace#
```

Además, podemos ver al listar los paquetes instalados que `matplotlib:3.9.1` ha sido instalado ya que lo habíamos incluido en nuestro `requirements.txt`:

```
root@3034e1e2b2e0:/workspace# pip list
Package           Version
-----
contourpy         1.2.1
cyclor            0.12.1
fonttools         4.53.1
kiwisolver        1.4.5
matplotlib        3.9.1
numpy             2.0.1
packaging         24.1
pillow            10.4.0
pip               24.2
pyparsing         3.1.2
python-dateutil   2.9.0.post0
setuptools        72.1.0
six               1.16.0
wheel             0.43.0
```

Un ejemplo de uso de esta librería:



Como hemos montado nuestra carpeta como volumen, al salir del contenedor con `ctrl+P+Q` tenemos en nuestra carpeta los siguientes ficheros:

```
jc@iJC > ls -l datos
total 48
```

```
-rw-r--r--  1 jc  staff   16669   2 ago 12:54 plot.png
-rw-r--r--  1 jc  staff    137   2 ago 12:53 script.py
```

Aunque pueda parecer complejo el empleo de Docker y los volúmenes al principio, ésta es una herramienta muy potente, ya que nos permite replicar en cualquier máquina o sistema operativo un entorno de trabajo muy específico (ya no tenéis excusa de que no habéis podido instalar alguna herramienta). Esta herramienta ganará importancia a lo largo de vuestra carrera profesional pues muchos servidores trabajan con esta tecnología y no sólo para aplicaciones, sino también para experimentación en Inteligencia Artificial.

3. Introducción a Python

En Algoritmia y Optimización trabajaremos con Python, por lo que a continuación se detalla la sintaxis básica del lenguaje.

3.1. Variables

Las variables en Python se utilizan para almacenar valores. Pueden ser del tipo entero (`int`), cadena (`str`), flotante (`float`) o booleano (`bool`). Para declarar una variable, simplemente asigna un valor a ella utilizando el operador de asignación (`=`).

```
x = 10    # Declarando una variable entera con valor 10
edad = "20" # Variable de tipo cadena
pi = 3.14  # Variable flotante con valor 3,14
esto_es_falso = False # Variable booleana
```

3.2. Tipos de datos

- **Enteros (`int`):** Son números enteros sin decimales.

```
x = 10    # Valor entero positivo
y = -5    # Valor entero negativo
```

- **Flotantes (`float`):** Son números con decimales.

```
pi = 3.14  # Valor flotante positivo
```

3.3. Cadenas

Las cadenas en Python se representan entre comillas simples o dobles. Utilizamos la función `print()` para imprimir valores en pantalla.

```
nombre = "Juan"    # Cadena de texto sin comillas
edad = '20'        # Cadena con comillas simples
```

3.4. Operadores

- **De suma (+):** Utilizado para sumar dos valores.

```
a = 5 + 3
b = "Hola, " + "Juan"
print(a)          # Resultado en pantalla: 8
print(b)          # Resultado en pantalla: Hola, Juan
```

- **De resta (-):** Utilizado para restar un valor de otro.

```
a = 5 - 3
print(a)          # Resultado en pantalla: 2
```

3.5. Estructuras de control

Estas estructuras nos permiten controlar el flujo del programa basándonos en ciertas condiciones.

- **Condición (if):** Utilizada para ejecutar un bloque de código si una condición es verdadera.

```
edad = 18

if (edad >= 18):
    print("Puedes votar")
```

3.6. Estructuras de repetición

- **Bucle (for):** Utilizado para ejecutar un bloque de código varias veces con valores en una lista o rango.

```
frutas = ['manzana', 'plátano', 'naranja']

for fruta in frutas:
    print(fruta)
```

- **Bucle (while):** Utilizado para ejecutar un bloque de código mientras cierta condición sea verdadera.

```
i = 0

while i < 10:
    print(i)
    i += 1
```

4. Introducción a Matplotlib

En sucesivas prácticas tendréis que mostrar los resultados de vuestros experimentos a través de gráficas para la cuales será necesario emplear Matplotlib. A continuación se describen funciones básicas de la librería.

4.1. Crear un gráfico básico

La forma más básica de crear un gráfico es con la función `plot()`.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.plot(x, y)
plt.savefig("plot.png")
```

Esto creará un gráfico de líneas simple.

4.2. Personalizar el gráfico

Puedes personalizar el gráfico añadiendo etiquetas y títulos.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.plot(x, y)
plt.title('Simple Line Plot')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.savefig("custom_plot.png")
```

4.3. Crear múltiples gráficos

Puedes crear múltiples gráficos usando la función `subplots()`.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [2, 3, 5, 7, 11]

plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
plt.plot(x, y1)

plt.subplot(1, 2, 2)
plt.plot(x, y2)

# bbox_inches="tight" hace que no
# tenga bordes la figura, prueba a no ponerlo
plt.savefig("subplot.png", bbox_inches="tight")
```

4.4. Gráficos 3D

Puedes crear gráficos 3D usando la función `plot3D()`, aunque esto no lo emplearemos en la asignatura.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = [1, 2, 3]
y = [4, 5, 6]
z = [7, 8, 9]

ax.plot(x, y, z)
plt.savefig("3d_plot.png")
```

Para finalizar esta introducción al paquete de Python: Matplotlib, crea gráficas en las que varíen los tipos de líneas, se asocien etiquetas a las funciones y aparezcan leyendas.

Si tienes alguna duda acude a la documentación oficial de Matplotlib.