# OD - Lab 1: Docker (Intro)

Link to exercise: https://docs.google.com/document/d/1W8xIGOcFI2HB1tYMJm5r95EEBl_aym0nb4Kl0iKHdtw/edit#

## Exercise 1

In this exercise, your task is to create a container that runs the above-mentioned hello-world image.

```
docker run hello-world
```

## Exercise 2

**Your tasks:**

1. Create a Dockerfile that prints the following message to the screen: ‘Hello, World! 42.’.
2. Build the above created Dockerfile. **Remember** to give the build a tag.
3. Run the above builded image.

The exercise will return the following message: "Hello, World! 42."

1. Creating the  "Dockerfile"

```
FROM ubuntu

RUN echo “Some text” > helloWorld.txt

CMD cat helloWorld.txt
```

2. Build the Dockerfile with a tag

```
docker build -t exercise2 .
```

3. Run the above builded image

```
docker run exercise2
```

## Exercise 3

**Included files:**

- DockerfileL1E3

  ```
  FROM ubuntu

  COPY L1E3/ /home/

  CMD bash
  ```

- print.py

  ```
  result = 2 * 100
  print(result)
  ```

**Your tasks:**

1. Build the included Dockerfile. **Remember** to give the build a tag.
2. Run the above builded image. **Remember** to run it in a way, which makes it possible for you to enter the container.
3. Install python3 and run the included print.py file.
4. Edit the Dockerfile, so python3 is installed beforehand and runs the Python script.

The exercise will return the following message:200

1. Building the included "Dockerfile"

```
docker build -t exercise3 -f DockerfileL1E3 .
```

2. Run the above builded image in a way, which makes it possible for you to enter the container.

```
docker run -it exercise3 bash # check the directory by typing "ls"
```

3. Installing python3 and running the included python file

```
$ apt update
$ apt install python3
$ cd /home
$ python3 print.py
```

4. Editing the DockerfileL1E3, so python3 is installed beforehand and runs the python script

```
#FROM ubuntu

#COPY L1E3/ /home/

#CMD bash

FROM python:3

COPY L1E3/ /./

CMD ["print.py"]

ENTRYPOINT ["python3"]
```

# Exercise 4

**Included files:**

- DockerfileL1E4

```
FROM php:7.2-apache

COPY ./L1E4 /var/www/html

RUN echo 'ServerName localhost' >> /etc/apache2/apache2.conf
```

- index.php

```
<html>
  <head>
    <title>
      PHP Test
    </title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>' ?>
  </body>
</html>
```

**Your tasks:**

1. Examine the included Dockerfile. Place your files, so it complies to the structure dictated by the Dockerfile.
2. Build the included Dockerfile. **Remember** to give the build a tag.
3. Run the above builded image. **Remember** to expose the port.
4. Visit the forwarded port through a browser on your host machine.

The exercise will return the following message in the terminal:

```
[Thu Jul 04 13:29:48.343170 2019] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.25 (Debian) PHP/7.2.19 configured -- resuming r
[Thu Jul 04 13:29:48.343489 2019] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'

172.17.0.1 - - [04/Jul/2019:13:20:20 +0000] "GET / HTTP/1.1" 200 370 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWeb
```

1. Examing the included Dockerfile and placing the files, so it complies to the structure dictated by the Dockerfile.
2. Building the included Dockerfile

```
docker build -t exercise4 -f DockerfileL1E4 .
```

3. Running the above builded image, and exposing the port

```
docker run -p 80:80 exercise4
```

4. Visiting the forwarded port through a browser on the host machine

```
localhost:80 # Or just "localhost"
```

Finish off by checking the terminal to assure the message is matching with the expected message

# Exercise 5

**Your tasks:**

1. Create a Python script that makes some form of output that is printed to the terminal.
    a. Inspiration: Make a for-loop that counts. Use the time.sleep(int count) method to make the executing Thread sleep.
2. Create a Dockerfile that copies the script to the container, installs Python and runs the script.

The exercise will return the following message: `SCRIPT-OUTPUT`

1. Creating a python script that makes some form of output that's printed on the terminal
    a. counting.py

```
import time

for x in range(11):
  print(x)
  time.sleep(1)
```

2. Creating a Dockerfile that copies the script to the container, installs Python and runs the script.
    a. Dockerfile

```
FROM python:3.

COPY script/ /./
```

```
CMD ["script.py"]

ENTRYPOINT ["python3"]
```

# Exercise 6

In this exercise the student is going to explain different concepts to/with the group. It is important to grasp the concepts of Docker and some base-knowledge.

**Your tasks:**

1. **Explain what a Dockerfile is.**
    a. How does it work?
        i. "A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession."
    b. Which instructions can be used?
        i. FROM, RUN, CMD, ENTRYPOINT, WORKDIR, COPY, ADD, EXPOSE, LABEL
    c. What do the above-mentioned instructions do?
        i. `FROM` = A FROM command allows you to create a base image such as an operating system, a programming language, etc. All the instructions executed after this command take place on this base image
        ii. `RUN` = A RUN instruction is used to run specified commands. You can use several RUN instructions to run different commands. But it is an efficient approach to combine all the RUN instructions into a single one as each RUN command creates a new cache layer
        iii. `CMD` = If you want to run a docker container by specifying a default command that gets executed for all the containers of that image by default, you can use a CMD command
        iv. `ENTRYPOINT` = The difference between ENTRYPOINT and CMD is that, if you try to specify default arguments in the docker run command, it will not ignore the ENTRYPOINT arguments.
        v. `WORKDIR` = You can specify your working directory inside the container using the instruction.
        vi. `COPY` = This instruction allows you to copy a directory from your local machine to the docker container.
        vii. `ADD` = Similar to COPY instruction, you can use ADD to copy files and folders from your local machine to docker containers. However, ADD also allows you to copy files from a URL as well as a tar file.
        viii. `EXPOSE` = The EXPOSE instruction inside the dockerfile informs that the container is listening to the specified port in the network. The default protocol is TCP.
        ix. `LABEL` = You can use a LABEL instruction to add description or meta data for a docker image. Its a key−value pair.
2. **Explain how a Docker image is built.**
    a. Which options do you know?
        i. "docker build", -t (TAG), -f (FILENAME), PATH-TO-DOCKERFILE
    b. What are they used for?
        i. `docker build` = the base command for building images
        ii. `-t` (TAG) = a tag used after the base command to give the image a name
        iii. `-f` (FILENAME) = a tag that can be used after the base command to specify a dockerfile thats named something else than "Dockerfile"
        iv. `PATH-TO-DOCKERFILE` = To specify the path to the dockerfile
3. **Explain how a Docker container is runned.**
    a. Which options do you know?
        i. The HyperVisor spins up a virtual environment (the container) and boots the image up and listens to the entrypoint.
        ii. "docker run", -it, -p, DOCKER-IMAGE, argument

b. What are they used for?

i. `docker run` = the base command for running images

ii. `-it` = a tag used after the base command to create interactive processes

iii. `-p` = is the option to expose ports (-p <HOST-PORT>:<CONTAINER-PORT>)

iv. `DOCKER-IMAGE` = the name for the docker image that should be runned

v. `argument` = makes it possible for us to enter the container and interact with it under runtime. (e.g "bash")