Working with Python Scripts

Introduction

Welcome to your first lab on fixing problems in Python. In this lab, you'll first have to fix an incorrect Python script. This includes:

- Fixing the file permissions to make it executable.
- Fixing a bug in the code.

After that, you'll write your own Python module and use it from the original script.

Fix file permissions

We have a python script ready for you. From your home directory (~), use the following command to navigate to the scripts directory:

```
cd scripts
```

List the files to find the script using the following command:

```
ls
```

To view the contents of this file, enter the following command:

```
cat health_checks.py
```

This Python file consists of a script to check disk and cpu usage. You can see shutil and psutil modules are imported here.

The *shutil* module offers a number of high-level operations on files and collections of files. In particular, it provides functions that support file copying and removal. It comes under Python's standard utility modules. *disk_usage()* method is used to get disk usage statistics of the given path. This method returns a named tuple with the attributes *total*, *used*, and *free*. The *total* attribute represents the total amount of space, the *used* attribute represents the amount of used space, and the *free* attribute represents the amount of available space, in bytes.

psutil (Python system and process utilities) is a cross-platform library for retrieving information on the processes currently running and system utilization (CPU, memory, disks, network, sensors) in Python. It's useful mainly for system monitoring, profiling, limiting process resources, and managing running processes. *cpu_percent()* returns a float showing the current system-wide CPU use as a percentage. When the interval is 0.0 or None (default), the function compares process times to system CPU times elapsed since the last call, returning immediately (non-blocking). That means that the first time it's called it will return a meaningful 0.0 value. When the interval is > 0.0, the function compares process times to system CPU times elapsed before and after the interval (blocking).

This script begins with a line containing the #! character combination, which is commonly called hash bang or shebang and continues with the path to the interpreter.

#!/usr/bin/env python3 uses the operating system env command, which locates and executes Python by searching the PATH environment variable. Unlike Windows, the Python interpreter is usually already in the $PATH variable on linux, so you don't have to add it.

Now that you understand what the script does, and the functions within it, let's run the Python file using the following command:

```
./health_checks.py
```

We got a permission denied error.

```
-bash: ./health_checks.py: Permission denied
```

This is because the above command tries to run your script directly as a program. The program is parsed by the interpreter specified in the first line of the script, i.e. shebang. If the kernel finds that the first two bytes are #! it uses the rest of the line as an interpreter and passes the file as an argument. So, to do this, the file needs to have execute permission.

To run this file, we need it to have execute permission (x). Let's update the file permissions and then try running the file. Use the following command to add execute permission to the file:

```
sudo chmod +x health_checks.py
```

Now try running the file again by using the following command:

```
./health_checks.py
```

This time, the output shows "ERROR".

```
ERROR!
```

Debug the issue

The Python script returns ERROR only if there's not enough disk usage or CPU usage. Try to debug this issue.

Hint: The problem is that the function **check_cpu_usage** should return **true** if the CPU usage is less than 75%, but in this case, it returns false.
Use a nano editor to open the file health_checks.py.

```
nano health_checks.py
```

Make the necessary changes now. And once the changes are done, save the file by clicking Ctrl-o, enter key and Ctrl-x.

Once you have debugged the issue, try running the file again by using the command:

```
./health_checks.py
```

This time, if the script is correct, the output should be "Everything ok".

```
Everything ok
```
Congratulations! You fixed the script!

Create a new Python module

In this section, you are going to write a Python module. A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

The module you are going to write will be used to test the network connections. We will guide you step by step through this process. Throughout the lab, we will refer to this module as the network module.

Let's start writing this network module. Since, the network module will check whether the network is correctly configured on the computer, we will use the requests module.

What is the requests module?

Requests is a Python module that you can use to send all kinds of HTTP requests. It's an easy-to-use library with a lot of features ranging from passing parameters in URLs to sending custom headers and SSL verification. You can add headers, form data, multi-part files, and parameters with simple Python dictionaries.You can then access the response data using the same request.

To use the requests module, you first need to install it. Use the following command to install the request module. If you receive any prompts, continue by clicking **Y**.

```
sudo apt install python3-requests
```

Create a file named `network.py`. The file should be created in the same directory as `health_checks.py`, i.e., **scripts**. If you are not present in the scripts directory, navigate to the scripts directory first and then create the file.

```
cd ~/scripts
```

Use nano editor to create a new file `network.py`:

```
nano network.py
```

Add a shebang line to define where the interpreter is located. In this case, the shebang line would be /usr/bin/env python3.

```
#!/usr/bin/env python3
```

Import the request module into the file using the import statements.

```
import requests
```

To check whether the local host is correctly configured, we use the socket module.

Now, import the socket module.

```
import socket
```

Next, write a function **check_localhost**, which checks whether the local host is correctly configured. We do this by calling the *gethostbyname* within the function.

```
localhost = socket.gethostbyname('localhost')
```

The above function translates a host name to IPv4 address format. Pass the parameter **localhost** to the function *gethostbyname*. The result for this function should be **127.0.0.1**.

Edit the function **check_localhost** so that it returns true if the function returns **127.0.0.1**.

Now, we will write another function called **check_connectivity**. This checks whether the computer can make successful calls to the internet.

A request is when you ping a website for information. The **Requests** library is designed for this task. You will use the request module for this, and call the GET method by passing *http://www.google.com* as the parameter.

```
request = requests.get("http://www.google.com")
```

This returns the website's status code. This status code is an integer value. Now, assign the result to a response variable and check the *status_code* attribute of that variable. It should return **200**.

Edit the function **check_connectivity** so that it returns true if the function returns **200** status_code.

Once you have finished editing the file, press Ctrl-o, Enter, and Ctrl-x to exit.

Use the Python module

Now, you're going to re-edit the file **health_checks.py** to make it call the checks in the network module. For this, you will need to import the module into health_checks.py script.

To do this, open the script `health_checks.py`

```
nano health_checks.py
```

Now import network module at the beginning of the file.

```
from network import *
```

Call the checks to the network module by adding an elif clause after the if clause in the script health_checks.py.

Replace the **else** part with an elif clause.

```
elif check_localhost() and check_connectivity():
    print("Everything ok")
```

Add an **else** part at the end of the file.

```
else:
    print("Network checks failed")
```

Once you have completed editing the file, press Ctrl-o, Enter, and Ctrl-x to exit.

Now, run the file.

```
./health_checks.py
```

It should return "Everything ok".

```
Everything ok
```