Working with Python Scripts

1 hour 30 minutesFree

Rate Lab

**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.

You'll have 90 minutes to complete this lab.

Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

**Note:** For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

Start Lab

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:



**Accessing the virtual machine**

Please find one of the three relevant options below based on your device's operating system.

**Note:** Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you’ll be able to practice through the labs.

Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM’s External IP address to connect.

**Download your PPK key file**

You can download the VM’s private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.



**Connect to your VM using SSH and PuTTY**

1. You can download Putty from [here](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)
2. In the **Host Name (or IP address)** box, enter username@external\_ip\_address.

**Note:** Replace **username** and **external\_ip\_address** with values provided in the lab.



1. In the **Category** list, expand **SSH**.
2. Click **Auth** (don’t expand it).
3. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
4. Click on the **Open** button.

**Note:** PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



1. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

**Common issues**

If PuTTY fails to connect to your Linux VM, verify that:

* You entered **<username>**@**<external ip address>** in PuTTY.
* You downloaded the fresh new PPK file for this lab from Qwiklabs.
* You are using the downloaded PPK file in PuTTY.

Option 2: OSX and Linux users: Connecting to your VM via SSH

**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to the VM using the local Terminal application**

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
   * To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
   * To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

**Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

chmod 600 ~/Downloads/qwikLABS-XXXXX.pem

ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address



Option 3: Chrome OS users: Connecting to your VM via SSH

**Note:** Make sure you are not in **Incognito/Private mode** while launching the application.

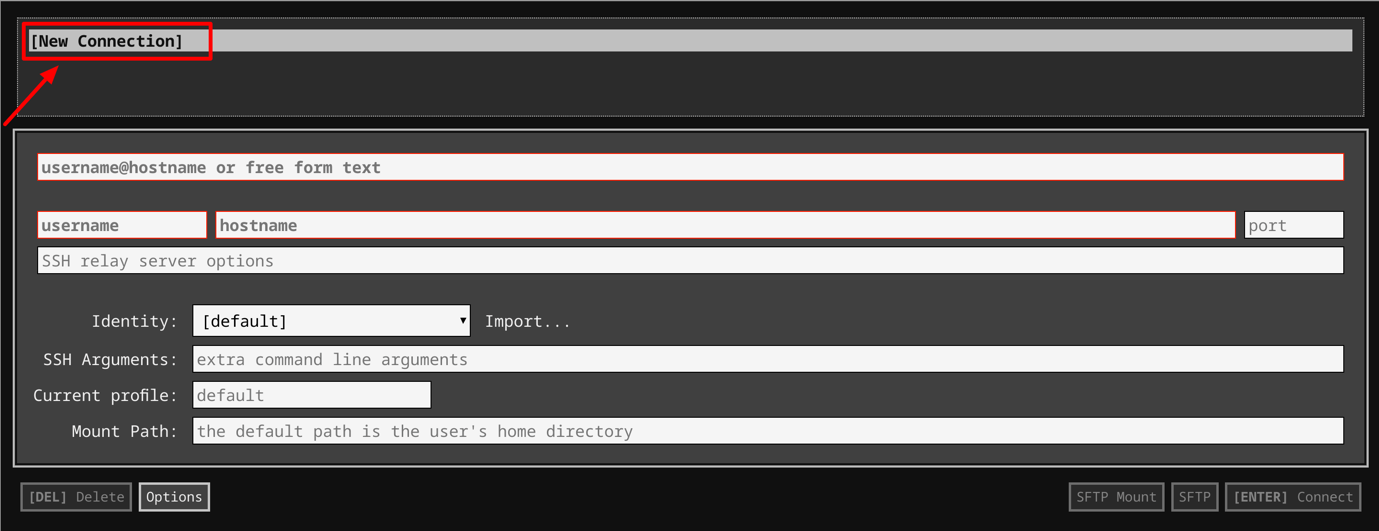
**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to your VM**

1. Add Secure Shell from [here](https://chrome.google.com/webstore/detail/secure-shell-app/pnhechapfaindjhompbnflcldabbghjo) to your Chrome browser.
2. Open the Secure Shell app and click on **[New Connection]**.



1. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.



1. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

**Note:** If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

1. Once your key is uploaded, click on the **[ENTER] Connect** button below.



1. For any prompts, type **yes** to continue.
2. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

**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 heath\_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.

A picture containing object

Description automatically generated

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".

A close up of a logo

Description automatically generated

Click *Check my progress* to verify the objective.

Fix file permissions

Check my progress

**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".



Congratulations! You fixed the script!

Click *Check my progress* to verify the objective.

Debug the issue

Check my progress

**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.

Click *Check my progress* to verify the objective.

Write a python module

Check my progress

**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".



Click *Check my progress* to verify the objective.

Use the python module

Check my progress

**Congratulations!**

You've successfully fixed an incorrect Python script, created a new Python module, and used it in the original script. These are common tasks for IT support roles, and mastering the skills you've practiced in this lab will greatly help you in your IT career. You can now close the RDP/SSH window, manually end the lab, and continue onto the next module. This lab will automatically end when the time runs out.

# Handling Files

1 hour 30 minutesFree

Rate Lab

## Introduction

For this lab, imagine you are an IT Specialist at a medium-sized company. The Human Resources Department at your company wants you to find out how many people are in each department. You need to write a Python script that reads a CSV file containing a list of the employees in the organization, counts how many people are in each department, and then generates a report using this information. The output of this script will be a plain text file. We will guide you through each step of the lab.

You'll have 90 minutes to complete this lab.

### Start the lab

You'll need to start the lab before you can access the materials in the virtual machine OS. To do this, click the green “Start Lab” button at the top of the screen.

**Note:** For this lab you are going to access the **Linux VM** through your **local SSH Client**, and not use the **Google Console** (**Open GCP Console** button is not available for this lab).

Start Lab

After you click the “Start Lab” button, you will see all the SSH connection details on the left-hand side of your screen. You should have a screen that looks like this:



## Accessing the virtual machine

Please find one of the three relevant options below based on your device's operating system.

**Note:** Working with Qwiklabs may be similar to the work you'd perform as an **IT Support Specialist**; you'll be interfacing with a cutting-edge technology that requires multiple steps to access, and perhaps healthy doses of patience and persistence(!). You'll also be using **SSH** to enter the labs -- a critical skill in IT Support that you’ll be able to practice through the labs.

### Option 1: Windows Users: Connecting to your VM

In this section, you will use the PuTTY Secure Shell (SSH) client and your VM’s External IP address to connect.

**Download your PPK key file**

You can download the VM’s private key file in the PuTTY-compatible **PPK** format from the Qwiklabs Start Lab page. Click on **Download PPK**.



**Connect to your VM using SSH and PuTTY**

1. You can download Putty from [here](https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe)
2. In the **Host Name (or IP address)** box, enter username@external\_ip\_address.

**Note:** Replace **username** and **external\_ip\_address** with values provided in the lab.



1. In the **Category** list, expand **SSH**.
2. Click **Auth** (don’t expand it).
3. In the **Private key file for authentication** box, browse to the PPK file that you downloaded and double-click it.
4. Click on the **Open** button.

**Note:** PPK file is to be imported into PuTTY tool using the Browse option available in it. It should not be opened directly but only to be used in PuTTY.



1. Click **Yes** when prompted to allow a first connection to this remote SSH server. Because you are using a key pair for authentication, you will not be prompted for a password.

**Common issues**

If PuTTY fails to connect to your Linux VM, verify that:

* You entered **<username>**@**<external ip address>** in PuTTY.
* You downloaded the fresh new PPK file for this lab from Qwiklabs.
* You are using the downloaded PPK file in PuTTY.

### Option 2: OSX and Linux users: Connecting to your VM via SSH

**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to the VM using the local Terminal application**

A **terminal** is a program which provides a **text-based interface for typing commands**. Here you will use your terminal as an SSH client to connect with lab provided Linux VM.

1. Open the Terminal application.
   * To open the terminal in Linux use the shortcut key **Ctrl+Alt+t**.
   * To open terminal in **Mac** (OSX) enter **cmd + space** and search for **terminal**.
2. Enter the following commands.

**Note:** Substitute the **path/filename for the PEM** file you downloaded, **username** and **External IP Address**.

You will most likely find the PEM file in **Downloads**. If you have not changed the download settings of your system, then the path of the PEM key will be **~/Downloads/qwikLABS-XXXXX.pem**

chmod 600 ~/Downloads/qwikLABS-XXXXX.pem

ssh -i ~/Downloads/qwikLABS-XXXXX.pem username@External Ip Address



### Option 3: Chrome OS users: Connecting to your VM via SSH

**Note:** Make sure you are not in **Incognito/Private mode** while launching the application.

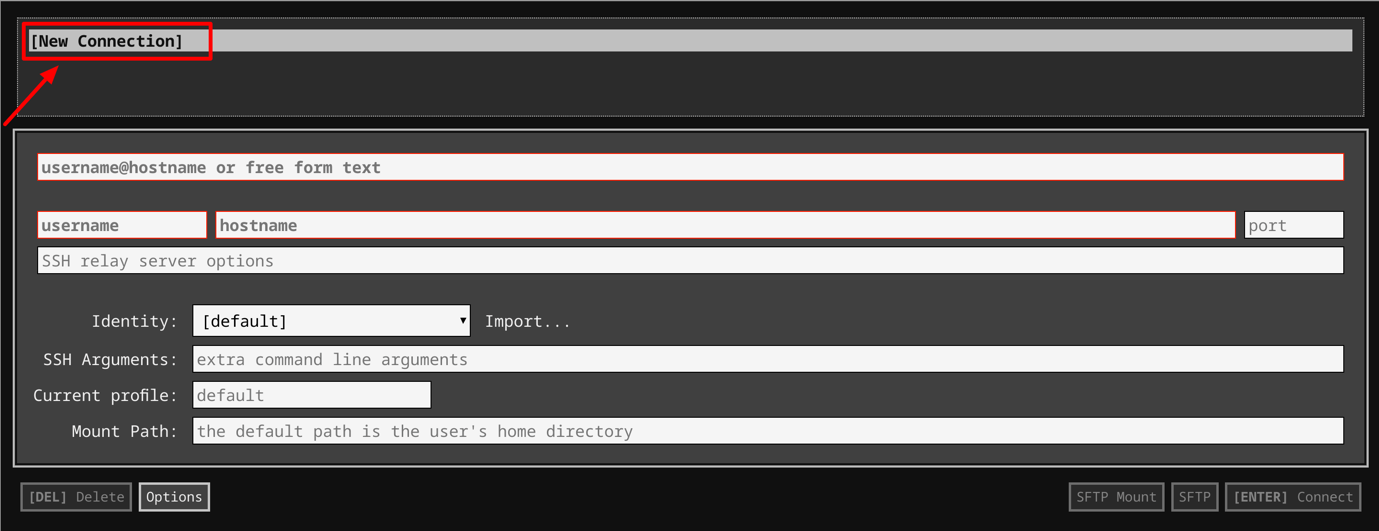
**Download your VM’s private key file.**

You can download the private key file in PEM format from the Qwiklabs Start Lab page. Click on **Download PEM**.



**Connect to your VM**

1. Add Secure Shell from [here](https://chrome.google.com/webstore/detail/secure-shell-app/pnhechapfaindjhompbnflcldabbghjo) to your Chrome browser.
2. Open the Secure Shell app and click on **[New Connection]**.



1. In the **username** section, enter the username given in the Connection Details Panel of the lab. And for the **hostname** section, enter the external IP of your VM instance that is mentioned in the Connection Details Panel of the lab.



1. In the **Identity** section, import the downloaded PEM key by clicking on the **Import…** button beside the field. Choose your PEM key and click on the **OPEN** button.

**Note:** If the key is still not available after importing it, refresh the application, and select it from the **Identity** drop-down menu.

1. Once your key is uploaded, click on the **[ENTER] Connect** button below.



1. For any prompts, type **yes** to continue.
2. You have now successfully connected to your Linux VM.

You're now ready to continue with the lab!

## Prerequisites

We have created the employee list for you. Navigate to the data directory using the following command:

cd data

To find the data, list the files using the following command:

ls

You can now see a file called **employees.csv**, where you will find your data. You can also see a directory called scripts. We will write the python script in this directory.

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

cat employees.csv

Let's start by writing the script. You will write this python script in the scripts directory. Go to the scripts directory by using the following command:

cd ~/scripts

Create a file named **generate\_report.py** using the following command:

nano generate\_report.py

You will write your python script in this **generate\_report.py** file. 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. If the kernel finds that the first two bytes are #! then it uses the rest of the line as an interpreter and passes the file as an argument. We will use the following shebang in this script:

#!/usr/bin/env python3

## Convert employee data to dictionary

The goal of the script is to read the CSV file and generate a report with the total number of people in each department. To achieve this, we will divide the script into three functions.

Let's start with the first function: read\_employees(). This function receives a CSV file as a parameter and returns a list of dictionaries from that file. For this, we will use the CSV module.

The CSV module uses classes to read and write tabular data in CSV format. The CSV library allows us to both read from and write to CSV files.

Now, import the CSV module.

import csv

Define the function read\_employees. This function takes file\_location (path to employees.csv) as a parameter.

def read\_employees(csv\_file\_location):

Open the CSV file by calling **open** and then **csv.DictReader**.

DictReader creates an object that operates like a regular reader (an object that iterates over lines in the given CSV file), but also maps the information it reads into a dictionary where keys are given by the optional fieldnames parameter. If we omit the fieldnames parameter, the values in the first row of the CSV file will be used as the keys. So, in this case, the first line of the CSV file has the keys and so there's no need to pass fieldnames as a parameter.

We also need to pass a dialect as a parameter to this function. There isn't a well-defined standard for comma-separated value files, so the parser needs to be flexible. Flexibility here means that there are many parameters to control how csv parses or writes data. Rather than passing each of these parameters to the reader and writer separately, we group them together conveniently into a dialect object.

Dialect classes can be registered by name so that callers of the CSV module don't need to know the parameter settings in advance. We will now register a dialect **empDialect**.

csv.register\_dialect('empDialect', skipinitialspace=True, strict=True)

The main purpose of this dialect is to remove any leading spaces while parsing the CSV file.

The function will look similar to:

employee\_file = csv.DictReader(open(csv\_file\_location), dialect = 'empDialect')

You now need to iterate over the CSV file that you opened, i.e., employee\_file. When you iterate over a CSV file, each iteration of the loop produces a dictionary from strings (key) to strings (value).

Append the dictionaries to an empty initialised list **employee\_list** as you iterate over the CSV file.

employee\_list = []

for data in employee\_file:

employee\_list.append(data)

Now return this list.

return employee\_list

To test the function, call the function and save it to a variable called employee\_list. Pass the path to employees.csv as a parameter to the function. Print the variable employee\_list to check whether it returns a list of dictionaries.

employee\_list = read\_employees('<file\_location>')

print(employee\_list)

Replace <file\_location> with the path to the employees.csv (this should look similar to the path /home/<username>/data/employees.csv). Replace <username> with the one mentioned in Connection Details Panel at left hand side.

Save the file by clicking Ctrl-o, Enter, and Ctrl-x.

For the file to run it needs 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:

chmod +x generate\_report.py

Now test the function by running the file using the following command:

./generate\_report.py

The list employees\_list within the script should return the list of dictionaries as shown below.

A close up of a white background

Description automatically generated

Click Check my progress to verify the objective.

Convert employee data to dictionary

Check my progress

**Note:**You can now remove the print statements once you get the desired output and have been assessed for this section.

## Process employee data

The second function process\_data() should now receive the list of dictionaries, i.e., employee\_list as a parameter and return a dictionary of **department:amount**.

Open the file generate\_report.py to define the function.

nano generate\_report.py

def process\_data(employee\_list):

This function needs to pass the employee\_list, received from the previous section, as a parameter to the function.

Now, initialize a new list called department\_list, iterate over employee\_list, and add only the departments into the **department\_list**.

department\_list = []

for employee\_data in employee\_list:

department\_list.append(employee\_data['Department'])

The department\_list should now have a redundant list of all the department names. We now have to remove the redundancy and return a dictionary. We will return this dicationary in the format **department:amount**, where amount is the number of employees in that particular department.

department\_data = {}

for department\_name in set(department\_list):

department\_data[department\_name] = department\_list.count(department\_name)

return department\_data

This uses the set() method, which converts iterable elements to distinct elements.

Now, call this function by passing the employee\_list from the previous section. Then, save the output in a variable called dictionary. Print the variable dictionary.

dictionary = process\_data(employee\_list)

print(dictionary)

Save the file by clicking Ctrl-o, Enter, and Ctrl-x.

Now test the function by running the file using the following command:

./generate\_report.py

This should return a dictionary in the format **department: amount**, as shown below.



Click Check my progress to verify the objective.

Process employee data

Check my progress

**Note:**You can now remove the print statements once you get the desired output and have been assessed for this section.

## Generate a report

Next, we will write the function write\_report. This function writes a dictionary of **department: amount** to a file.

The report should have the format:

<department1>: <amount1>

<department2>: <amount2>

Lets open **generate\_report.py** file to define the function.

nano generate\_report.py

def write\_report(dictionary, report\_file):

This function requires a dictionary, from the previous section, and report\_file, an output file to generate report, to both be passed as parameters.

You will use the open() function to open a file and return a corresponding file object. This function requires file path and file mode to be passed as parameters. The file mode is 'r' (reading) by default, so you should now explicitly pass 'w+' mode (open for reading and writing, overwriting a file) as a parameter.

Once you open the file for writing, iterate through the dictionary and use write() on the file to store the data.

with open(report\_file, "w+") as f:

for k in sorted(dictionary):

f.write(str(k)+':'+str(dictionary[k])+'\n')

f.close()

Now call the function write\_report() by passing a dictionary variable from the previous section and also passing a report\_file. The report\_file passed within this function should be similar to /home/<username>/data/report.txt. Replace <username> with the one mentioned in Connection Details Panel at left-hand side.

write\_report(dictionary, '<report\_file>')

Save the file by clicking Ctrl-o, Enter, and Ctrl-x.

Let's execute the script now.

./generate\_report.py

This script does not generate any output, but it creates a new file named **report.txt** within the **data** directory. This report.txt file should now have the count of people in each department.

Navigate to the data directory and list the files. You should see a new file named **report.txt**.

cd ~/data

ls

To view the generated report file, use the following command:

cat report.txt

The report file should be similar to the below image.

A picture containing bird, flower

Description automatically generated

Click Check my progress to verify the objective.

Generate a report

Check my progress

## Congratulations!

You successfully wrote a Python script that achieves two tasks. First, it reads a CSV file containing a list of the employees in the organization. Second, it generates a report of the number of people in each department in a plain text file.

Creating reports using Python is a very useful tool in IT support. You will likely complete similar tasks regularly throughout your career, so feel free to go through this lab more than once. Remember, practice makes perfect.