

# Handling Files | Qwiklabs

Qwiklabs

14-18 minutes

---

## 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.

## 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.

```
student-04-b7e1d2307a6@linux-instance:~/scripts$ ./generate_report.py
[{'Department': 'Development', 'Username': 'audrey', 'Full Name': 'Audrey Miller'}, {'Department': 'Sales', 'Username': 'ardeng', 'Full Name': 'Arden Garcia'}, {'Department': 'Human Resources', 'Username': 'baileyt', 'Full Name': 'Bailey Thomas'}, {'Department': 'IT infrastructure', 'Username': 'sousa', 'Full Name': 'Blake Sousa'}, {'Department': 'Marketing', 'Username': 'nguyen', 'Full Name': 'Cameron Nguyen'}, {'Department': 'Development', 'Username': 'greyc', 'Full Name': 'Charlie Grey'}, {'Department': 'User Experience Research', 'Username': 'chrisb', 'Full Name': 'Chris Black'}, {'Department': 'IT infrastructure', 'Username': 'silva', 'Full Name': 'Courtney Silva'}, {'Department': 'IT infrastructure', 'Username': 'darcy', 'Full Name': 'Darcy Johnsonn'}, {'Department': 'Development', 'Username': 'elliottl', 'Full Name': 'Elliot Lamb'}, {'Department': 'Sales', 'Username': 'halls', 'Full Name': 'Emery Halls'}, {'Department': 'Marketing', 'Username': 'flynn', 'Full Name': 'Flynn McMillan'}, {'Department': 'Human Resources', 'Username': 'harley', 'Full Name': 'Harley Klose'}, {'Department': 'Vendor operations', 'Username': 'jeannm', 'Full Name': 'Jean May Coy'}, {'Department': 'Sales', 'Username': 'kstev', 'Full Name': 'Kay Stevens'}, {'Department': 'User Experience Research', 'Username': 'lion', 'Full Name': 'Lio Nelson'}, {'Department': 'Vendor operations', 'Username': 'tillas', 'Full Name': 'Logan Tillas'}, {'Department': 'Development', 'Username': 'micah', 'Full Name': 'Micah Lopes'}, {'Department': 'IT infrastructure', 'Username': 'solm', 'Full Name': 'Sol Mansi'}]
```

**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 dictionary 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.

```
jcpstagingedit1690_student@linux-instance:~/scripts$ ./generate_report.py
{'Sales': 3, 'Marketing': 2, 'IT infrastructure': 4, 'Vendor operations': 2, 'User Experience Research': 2, 'Human Resources': 2, 'Development': 4}
```

**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**.

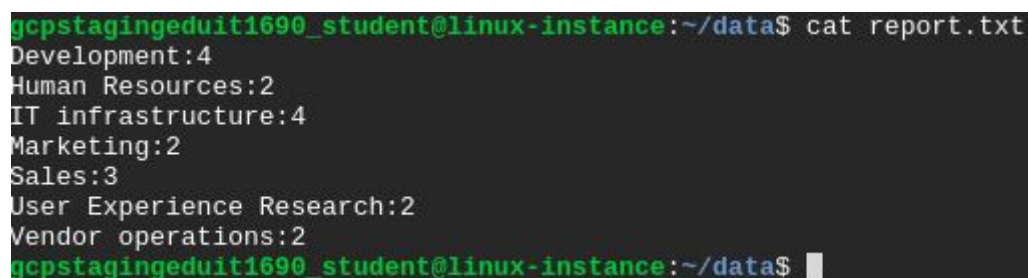
```
cd ~/data
```

```
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 terminal window screenshot showing the command `cat report.txt` being executed. The output lists departments and their counts: Development:4, Human Resources:2, IT infrastructure:4, Marketing:2, Sales:3, User Experience Research:2, and Vendor operations:2. The terminal prompt is `gcpstagingedit1690_student@linux-instance:~/data$`.

```
gcpstagingedit1690_student@linux-instance:~/data$ cat report.txt
Development:4
Human Resources:2
IT infrastructure:4
Marketing:2
Sales:3
User Experience Research:2
Vendor operations:2
gcpstagingedit1690_student@linux-instance:~/data$
```