Python in action

Well done! Everything that you've been learning and all your hard work have brought you to this point. You've learned how to write a Python script from the ground up to solve a complex problem. You've also practiced how to:

- Identify a problem statement.
- Research the tools needed to help solve the problem.
- Plan an approach or best strategy to figure out what needs to be done, how it's done, and how to structure the code.
- Write a script to solve the problem.
- Run and check that the code works the way it should.

Let's now take a look at an example of how you can apply this process in the real world. This reading will provide you with a real-life example and walk you through the application of each step in the process.

The problem and solution

Imagine this scenario: Every month, you are handed a spreadsheet with hundreds of new hires. You are asked to create user accounts for all of them on a Linux server. The formatting on the spreadsheet looks like this:

```
username,password,real_name

amanda,,Amanda Alonso

ian,,Ian Ortega

eugene,,Eugene Konya

[...]
```

Notice that the password field is empty for all the records. This means you need to generate random passwords for each user and then create their accounts. You also want to write the passwords that you generate back to a new CSV file so that you can tell the new employees their passwords.

This task isn't difficult, but it is time-intensive if you create passwords and accounts for the hundreds of new hires one by one. Your solution is to automate this task in Python.

The script

To automate the task of creating passwords and accounts for all of these new hires, the script should do the following:

- Read a list of new hires from **`users_in.csv`**.
- Generate random 16-character passwords for each user.
- Create each user account.

- Write the spreadsheet back to **`users_out.csv`** with the new passwords.

Your tools

To help organize all the data, create accounts for the new hires, and create passwords for each new user, you first need to import a few standard Python libraries.

```
import csv
```

This library helps read and write the CSV files.

```
import secrets
```

This helps generate random passwords for each user account.

```
import subprocess
```

This calls the **`useradd`** command, which creates and adds each user account.

```
from pathlib import Path   # to locate the data files
```

This library helps to locate the data files for each user account.

Getting started

After importing the libraries that help you execute your script, you need to get the current working directory and find the subdirectory where the CSV files are stored. Use **`cwd`** for "current working directory" and identify the path of the Python directory as a string:

```
cwd = Path.cwd() / "drive/MyDrive/Colab Notebooks"
```

Next, you use a **`with`** statement and an **`as`** keyword. The **`with`** statement helps with resource management, and the **`as`** keyword creates an alias for the resource you want to call. Consider the following code:

```
with open(cwd / "data/users_in.csv", "r") as file_input, open(cwd / "data/users_out.csv", "w") as file_output:
```

The CSV library takes care of reading and parsing the input from the file.

Next, you can use a **`DictReader`** object so that each row in the file is read into a **`dict()`** with the field names and values, like this: {"**`username`**": "**`amanda`**", "**`password`**": "", "**`real_name`**": "**`Amanda Alonso`**"}.

The following code is an example of how you use the **`DictReader`**:

```
    reader = csv.DictReader(file_input)
```

The input for the script is now complete! Now you need to set up the output. You create a **`DictWriter`** and use the same field names from the input, like so:

```
writer = csv.DictWriter(file_output,fieldnames=reader.fieldnames)
    writer.writeheader()
```

Now, you create a for loop to run through each record from the input file.

```
    for user in reader:
```

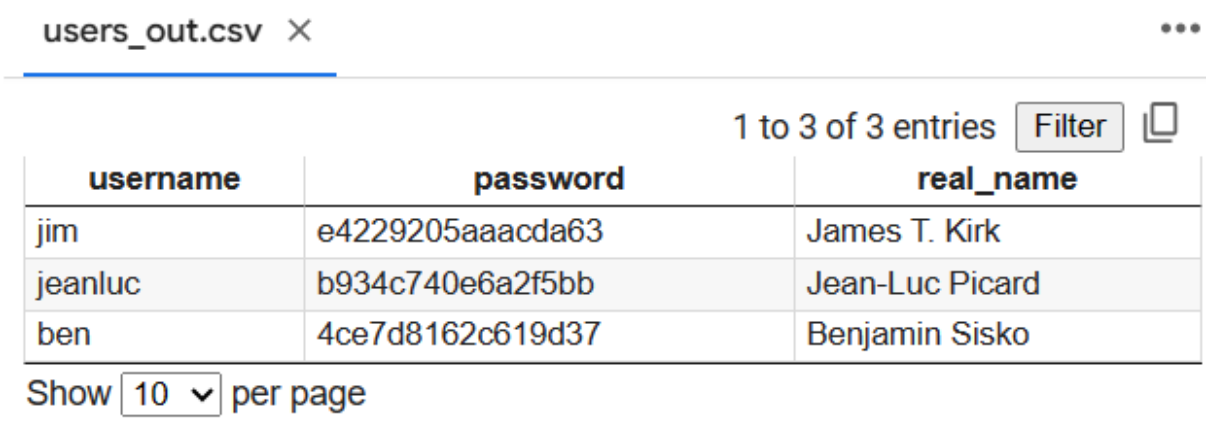
After the **`for`** loop, you use the **`secrets library`** that you imported at the beginning of the script to generate a random password of eight hex bytes, which equals 16 characters in total. Then, run the **`/sbin/useradd`** command to create each user. The **`check=True`** parameter causes the script to exit with a **`CalledProcessError`** if the command fails for any reason.

```
        user["password"] = secrets.token_hex(8)
        useradd_cmd = ["/sbin/useradd",
                       "-c", user["real_name"],
                       "-m",
                       "-G", "users",
                       "-p", user["password"],
                       user["username"]]
        subprocess.run(useradd_cmd, check=True)
```

Finally, you write the records back to the output file, including the passwords. When you run the code, the new user accounts and their passwords are generated into a new CSV file.

```
    writer.writerow(user)
```

After the script runs, the output CSV file should look something like this:



And there you have it! You've just saved yourself countless hours of creating hundreds of new employee accounts and passwords by creating a short, simple script to do the work for you.

Key takeaways

There are many real-world applications for using Python: creating programs, solving complex problems, simplifying and/or automating time-intensive tasks, and many more. But the process always remains the same—identifying a problem, coming up with a solution, determining the tools that help you achieve your solution, as well as the most significant part—writing the actual script! As you saw in this example, any time you have a repetitive task, think of using Python to automate that task. The programming skills you've learned can make the work you do in your IT job a lot faster and more efficient!