One of TAs has claimed that verifying certificates is too hard for students. We have created a challenge to prove him wrong. This challenge consists of four parts.

Before Starting

Before starting with the exercises, we suggest you download the `helper` script. It'll be a great help in solving the exercises and communicating with the grading server. You do not need to read or modify this script. Please tell the helper your SCIPER number by running:

```
$ wget -O helper.py https://com301.epfl.ch/static/files/helper.py
$ python3 helper.py config --sciper $YOUR_SCIPER
```

Self-signed certificate

Before verifying certificates, the TA want to see if you are capable of creating a certificate. The TA has decided on a a list of properties that your certificate should satisfy:

1. The certificate must have a 4096-bit RSA key.
2. The certificate must be issued for `$sciper.lab.spring.ch` where `$sciper` is your sciper number.
3. The certificate must be self-signed.

A self-signed certificate is a certificate that is signed by its owner instead of a CA. Root CAs use this process to create their certificate since there is no authority above them.

We are using X.509 certificates encoded in PEM files for this challenge. After creating your certificate `cert_p1.pem`, run the `helper` as follows to submit your results:

```
$ python3 helper.py part1 --cert cert_p1.pem
```

Extracting data from a certificate

After creating your certificate, the TA challenges you to extract data from a certificate. Download the certificate `cert_p2.pem` with the helper

```
$ python3 helper.py get cert-p2 -o cert_p2.pem
```

and answer the following questions.

1. Who owns the certificate?
2. Who issued the certificate?
3. What is the serial number of the certificate?

Run the `helper` as follows to submit your answers:

```
$ python3 helper.py part2 -o 'OWNER_CN' -i 'ISSUER_CN' -s 'SERIAL'
```

SSL in wild

You proved that you can work with certificates, but can you use them in practice? The TA has deployed a certificate authority (CA) and a web server and is serving sites with valid and invalid certificates. Can you find out which sites are using invalid certificates?

Download the incomplete Python script `skeleton3.py`. For now, it retrieves a list of 50 `host:port` pairs from the grading server. You still need to find out if these servers are using valid SSL certificates. Extend the method `homework6_part3` to gather the necessary information. Hint: you may want to use the `requests` library. Then run the complete script to submit your results to the grading server.

```
$ python3 skeleton3.py
```

Food for thought: You should not add the TA's certificate to your computer's global list of trusted CAs? Why is that?

Some browser do not let you download the CA certificate, in this case, `wget` *can be used:*

```
$ wget -O CA_cert.pem https://com301.epfl.ch/static/files/CA_cert.pem
```

Certificate pinning

Validation certificates is a fundamental skill, but the world is an evil place, and the TA is paranoid: he distrusts CAs and would much prefer to use self-signed certificates. The TA has paid attention in class, and realizes that accepting *all* self-signed certificates is a very bad idea. Therefore, he wants you to construct an application that only accepts his self-signed certificate, but not any others.

This procedure is called certificate pinning. To prove that you are up to the task, the TA has given you one self-signed certificate. Can you figure out which server is using it? You can retrieve the TAs certificate using the helper script:

```
$ python3 helper.py get cert-p4 -o cert_p4.pem
```

Download the incomplete Python script `skeleton4.py`. As in the previous exercise, complete the script to determine which server to trust. Then run the completed script to submit your results to the grading server.

```
$ python3 skeleton4.py
```