Immediately after that, when all system requirements are met, we use the command «`npm install cypress`» to install cypress.

Next, only two commands are needed to start the automation, which is written in the form of scripts on cypress.

`npm install - install all necessary libraries and dependencies that use scripts for automation.`

`npm test - run cypress to execute scripts.`

Next, as for running a similar configuration using docker, the most important thing here is to use the correct image. For this particular task, an image was selected from the official developer cypress cypress/ browsers:node18.6.0-chrome105-ff104. This is an image based on the Debian GNU/Linux 11 operating system. In which node js, the necessary libraries and dependencies, cypress and headless chromedriver are already installed to emulate the visual component, provided there is no physical output device. For subsequent recording, saving and sending various types of logs in the form of screenshots and videos to the storage, from where it is easily possible to view the results of automated actions.

The full Dockerfile will look like this:

```
FROM cypress/browsers:node18.6.0-chrome105-ff104
COPY . /cypress
WORKDIR /cypress
RUN npm i
ENTRYPOINT [ "npm", "test" ]
```

The essence of this docker file is that only scripts, a file with package dependencies are copied to the container.json and the cypress configuration in the form of a cypress.config file.js, and then automation is started using the npm test command.

Koua Lionel Ignace
(F. Skorina GSU, Gomel)
Scientific adviser **P. V. Bychkov**,
Ph.D. in Physics and Mathematics, associate professor

DEVELOPMENT OF SOFTWARE ENVIRONMENTS FOR MANAGING REMOTE NETWORK DEVICES

To run project, you will need python3 and pip packages installed on your system. Pip is a package management system that simplifies installation and management of software packages written in Python such as those found in the Python Package Index (PyPI). To install python3 and pip, open terminal and put following command:

```
sudo apt-get install python3 python3-pip
```

After that you would need to setup a new python virtual environment, so that all project's dependencies will be bound to project's directory and wouldn't be accessible outside that virtual environment. This step is not mandatory, so it might well be skipped. To create python virtual environment, run following command in terminal:

```
python3 -m venv /path/to/new/virtual/env
```

Now you need to install project's dependencies, they can be found in project's «REAME.md» file. To install all project's dependencies, run the following command in terminal:

```
pip3 install vymgmt pysftp schedule
```

Now, when all dependencies install you will need to adjust vymgmt library. By default, you cannot set a SSH-session timeout using library's methods. Default value is 30 seconds which is not enough to apply all rules. The timeout is crucial to faultless program working. To change timeout, you will need to find file «/usr/local/lib/python3.5/dist-packages/vymgmt/router.py». In that file find line «self.__conn = pxssh.pxssh(timeout=30)» and adjust value after «timeout=» to at least an hour (3600).

After that you will need to adjust configurational files. Each configurational file contains paths or credential. They have to be modified for correct work. Configurational files are json formatted and easy to read.

When all configs are modified it's time to set program working periods. Scheduling is done using schedule library, modification is easy to understand. To adjust periods "belgie_sync.py" has to be modified, example on in Figure 1.

```
schedule.every(10).minutes.do(job)
schedule.every().hour.do(job)
schedule.every().day.at("10:30").do(job)
schedule.every(5).to(10).minutes.do(job)
schedule.every().monday.do(job)
schedule.every().wednesday.at("13:15").do(job)
schedule.every().minute.at(":17").do(job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

Figure 1 – Example of schedule configuration