Python Project 2022

14. Juni 2022

1 Project Description

The Program developed is a simulator for sensor data. It should be capable of generating Values, storing them in a file, reading from the file and generating visual output via matplotlib.

2 Git

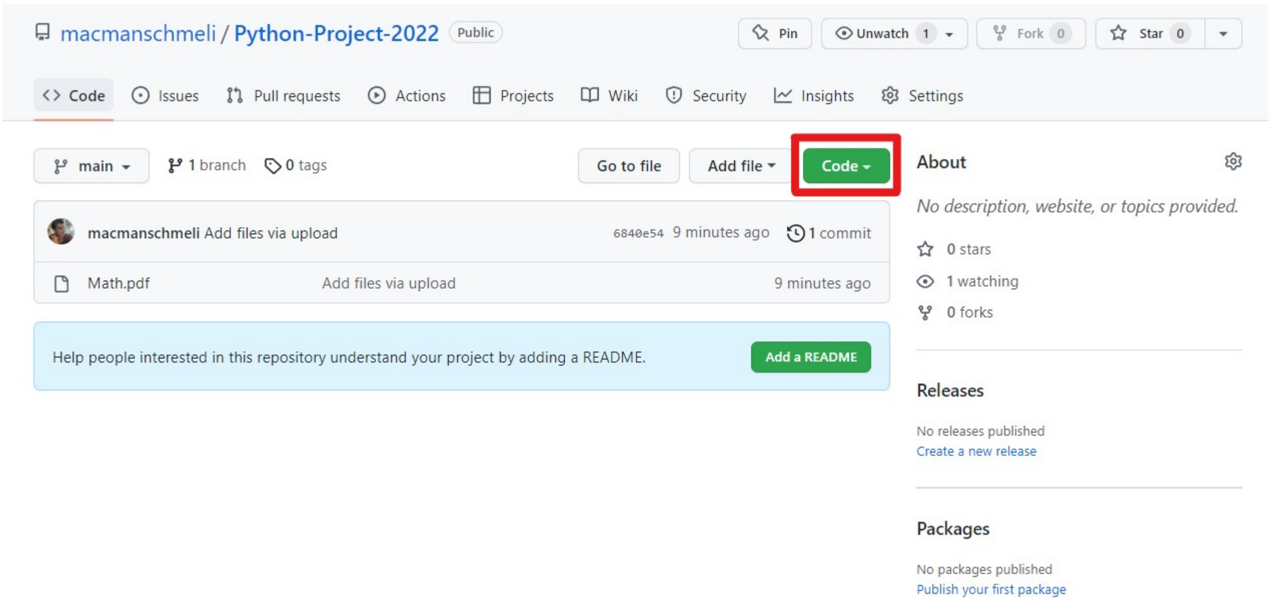
2.1 Setup

First you need to download a git client, on windows, a guide can be found through this link. After the install open a command window, on windows, you can do this by selecting the magnifying glass 🔍 in the windows taskbar and typing **cmd** and hitting enter. On Windows 11 you can do that by opening the file explorer, right click somewhere and click on **cmd**. Then type `> git --version` to check if the install was successful.

Next, you need to create a user on Github, which is free of charge. To do so, go to the website github.com and follow the instructions.

Next, to efficiently work with git you will have to create an ssh key. Therefore, you will have to install **ssh**. To do this on a windows pc, go to **Settings**, select **Apps** > **Apps & Features**, then select **Optional Features**. Scan the list, to see if OpenSSH is already installed. If not, at teh top of the page, select **Add a feature**, then find **OpenSSH Client**, click install and do the same with the **OpenSSH Server**. To check if the install was successful, or **ssh** is already installed, execute `> ssh -v localhost` in a terminal window. To generate an ssh-key pair run `> ssh-keygen` and hitting enter if asked. Next navigate to the directory, where the ssh keys are located. On windows this should be somewhere at C:/Users/name/.ssh. Sign in to Github and then go to "add github ssh key" and click on the green New SSH key button. Copy the contents of the `id_rsa.pub` file into the key field, give it a title and save it.

Then go to the github repository: Python Project 2022 and click on the green Code Button marked in the picture below.



macmanschmeli Add files via upload 6840e54 9 minutes ago 1 commit

Math.pdf Add files via upload 9 minutes ago

Help people interested in this repository understand your project by adding a README.

No description, website, or topics provided.

0 stars

1 watching

0 forks

No releases published

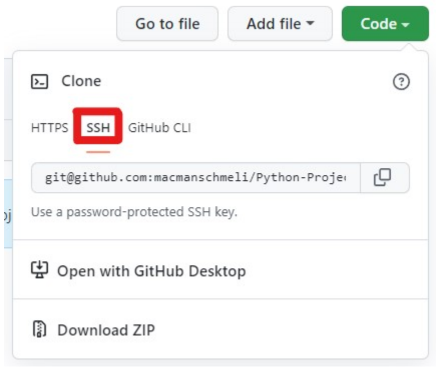
Create a new release

No packages published

Publish your first package

Select the **SSH** option, copy the link which should be

git@github.com:macmanschmeli/Python-Project-2022.git



```
git@github.com:macmanschmeli/Python-Proje
```

Use a password-protected SSH key.

Finally open a terminal window and type the command `> git clone` and then the copied link, such that as a whole, the command looks like this:

```
> git clone git@github.com:macmanschmeli/Python-Project-2022.git
```

Git will then download a local copy of the repository to your machine. But before you can start, you have to set your credentials to the ones associated with your git account. To do so run `> git config user.name "git name"` and `> git config user.email "git mail"` in the directory of the local repository.

2.2 Best Practices

Since in this exercise there will be more than one agent adding files to the repository and we want to avoid time consuming merge conflicts, before you start to work on the project, always open a terminal window, navigate to the repository directory and execute `> git pull` which fetches remote changes (aka files that are stored on the server and newer than the ones in your local copy of the repository).

Further after a development session execute `> git add file1 file2 ...` to add files to a commit (file1 and file2 beeing the files you want to commit) or alternatively `> git add .` to add all files to a commit, in a terminal window in the directory of the repository. Afterwards execute `> git commit -m "text that describes the commit"` to create a commit. To upload your changes to the remote execute `> git push` and wait for the process to finish. If `> git push` returns an error similar to `> upstream not set`, just follow the instructions and execute `> git branch -set-upstream-to origin/main` followed by another `> git push`.

If for some reason an error message tells you something along the lines of you cannot push because there were changes in the remote repository and you already have added your files to a commit and created a commit then just execute `> git pull` and then `> git push` and it should work fine. Else do not hesitate to contact me (***@ El Tigre***) on Slack.

3 Stage One

3.1 Argument Handling

In this stage, the argument handling for the simulator should be implemented. Argument handling is the way how arguments, that are given at the call of the program, are checked if they fit their purpose. As an example: a program `add`, that adds two numbers is called with the following call sequence:

```
> pyhton3 add.py <number_one> <number_two>
```

If the user calls the program with

```
> python3 add.py 1 2
```

everything is fine but what about:

```
> python3 add.py something 2
```

The adder will not know how the argument `something` should be interpreted, so internally the adder has to make sure that only valid arguments are actually processed, since otherwise errors may occur and the program can crash.

3.2 Reqirements

Our implementation of the program will be called in the following fashion:

```
> python3 sensor_generator.py <boundary_one> <boundary_two>
```

The program shall accept all numbers in the range of the float datatype in python. If the input values are out of range or not a number then the following error message shall be displayed and the program should terminate.

```
program terminated with errors!
input is not in range or not a number!
usage: ./sensor_generator.py <boundary_one> <boundary_two>
```

4 Stage Two

Now after the program is able to handle inputs correctly, actual functionality should be implemented. The task is to parse the inputs from the program and start generating random values in between the two given values. These values should be printed to the console in a csv style format.

4.1 CSV Style

CSV is a widely spread format for tables where colums are seperated by a "***;***"character and lines are seperated via a ***newline*** character aka "***\n***". As an example

```
name;time;value
pressuresensor;12:00;100
pressuresensor;12:01;110
```

will produce an output similar to the following table if opened with an application like Microsoft Excel or LibreOffice.

name	time	value
pressuresensor	12:00	100
pressuresensor	12:01	110

4.2 Requirements

The program should write the generated values in a CSV like format. The output must contain a header as follows:

```
name;time;value
```

The lines following the header should then contain the actual data associated with the header fields. For this stage it is sufficient to always assign the name "**sensor1**"to the name field. The time value should for now always start at 12:00 and progress in one minute steps as visible in the example provided below. For this stage the simulator should print a fixed amount of 20 datapoints. By datapoint we mean a value associated with one time value, for example one actual data line:

```
sensor1;12:00;100
```

So in the end the output must be a total of 21 lines.

4.3 Example

```
> python3 sensor_generator.py 100 120
name;time;value
sensor1;12:00;101
sensor1;12:01;111
sensor1;12:02;105
sensor1;12:03;119
sensor1;12:04;109
...
```