University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatics I – HS18

Exercise 10

Submission Details

- **Submission Format:** ZIP file containing the .py solution files of task 1, a copy of your local repository, and a .txt file containing the url of your GitHub repository.
- **Submission Deadline:** 12:00, Tuesday 4th December
- The name of the ZIP file must have the following format: firstname_lastname_studentidentificationnumber_info1_exercise_10.zip, e.g. *hans_muster_12345678_info1_exercise_10.zip*.
- Your ZIP file should contain the following files: `base64_encoder.py`, `task_1.py` and a PDF. **Do not rename the python files.**
- **Important:** Please follow the naming conventions very strictly, otherwise we will not be able to grade your submission.
- Please submit the assignment using OLAT. You may upload multiple solutions, but note that **only the last submission** will be evaluated.
- Your submissions will be tested on systems running **Python 3.7.X**. Make sure that your solutions work on this version.

1 Task: Base64 Encoder & Decoder (2 Points)

The goal of this task is to familiarize yourself with the concept of modules in Python. The exercise itself is quite simple, you will write an application that encodes a string using the base64 encoding scheme[1], sends it to a service on https://httpbin.org/ and then receives as answer the decoded string again. We have already provided a module that takes care of decoding a string:

```
1  from urllib import request
2
3
4  def decode_base64_str(encoded_str):
5      result = request.urlopen(f"https://httpbin.org/base64/{encoded_str}").
       read().decode('ascii')
6      return result
7
8
9  if __name__ == '__main__':
10     print(decode_base64_str("SFRUUEJJTiBpcyBhd2Vzb211"))
```

Listing 1: base64_decoder.py

You have the following instructions:

- Create a file `base64_encoder.py` that contains the encoding functionality. In that file, import `urlsafe_b64encode` from Python's standard module `base64` using the `from x import y` syntax. Do **not** rename the function when importing it. Then, create a function `encode_base64_str(string)` that takes a string and returns the base64-encoded string.
  Note: If you study the documentation of the `urlsafe_b64encode` function, you will see that it takes and returns bytes-like objects, not `str` objects. This means that you will have to parse the input string to a bytes-like object and the result of the operation back to a string. You can do that with `string.encode('ascii')` resp. `string.decode('ascii')`.
- Create another file `task_1.py`. In this file, you need to import two functions: The one you just created and the one we provided you. Import both of them with the `from x import y as z` syntax; rename the functions to `encode` resp. `decode`.
- Create a function `verify_encoding_decoding(string)` that takes a string, encodes it and decodes it again and checks whether the result is the same as the input string. Return the result of this comparison as a boolean value. You can check the example usage of this function below.

```
1  if __name__ == '__main__':
2      # verify_encoding_decoding should return True
3      assert verify_encoding_decoding("Try out any test string here!")
```

Listing 2: Sample usage.

Note: Be aware that the implementation of the decoder needs a working internet connection to function properly. If you try to execute the code without internet connection, you will get a timeout or an error.

Task: Create the files `base64_encoder.py` and `task_1.py` as indicated above.

[1] https://en.wikipedia.org/wiki/Base64

2 Task: Git (4 Points)

In this task you are going to create a repository on GitHub and modify it locally on you computer. Please hand in the link to your GitHub repository and a copy of your local Git repository. Make sure your GitHub repository is public and that you include all the files (also the hidden files) of your local repo.

We assume that your computer has a running version of Git installed. If not, please refer to the lecture slides for a link on how to do this or use Google to find instructions on how to install Git.

a) Create your first repository *(2 Points)*

First, create a Git repository on GitHub. Follow the instructions 1-6 in the following link `https://help.github.com/articles/create-a-repo/`. Give your repository any name and description, make it public, but do **not** initialize it with a README.

Next, clone the repository to your computer with the `git clone <repository-url>` command. Then do the following things:

- Create the file `hello_a.py` (For the entire exercise use the `touch` command to create files so you can better report what you have done)
- Add the file to the repository
- Commit the changes (always write meaningful commit messages to your commits)
- Push the changes to GitHub

b) Branches and Merging *(1 Point)*

Now you will practice the fundamentals of branching and merging. Please follow the instructions:

- Create a new branch with the name `develop`
- Switch to the new branch
- Create the file `hello_b.py`
- Commit the changes
- Add the statement `print("Hello Mrs. B")` to the file
- Commit the changes
- Push the new branch to GitHub [2]
- Checkout the master branch (the first branch in a project is always called master)
- Write the statement `print("Hello Mr. A")` to the file `hello_a.py`
- Commit the changes
- Now merge the develop branch into the master branch. (again, don't forget a meaningful commit message)
- Commit and push the changes

[2] The first time, if the branch does not already exist in your remote repository (in your case GitHub), you should use `git push -set-upstream origin develop` for pushing.

c) Conflict Resolution *(1 Point)*

In the previous task, you practised some basic branching and merging. When you merged develop into master, Git was capable to automatically merge the two branches. However, the auto merging of Git has its limits, for example when the same file has been modified in two separate branches in parallel. In this case, the user has to resolve the resulting conflict by hand; this is what you will practice in the current task.

- Create the file `hello_c.py` in the `master` and `develop` branch. Commit the changes.
- Write `print('Hello C')` to `hello_c.py` in the `master` branch and commit the changes
- Write `print('Hello c')` (with lower case c!) to `hello_c.py` in the `master` branch and commit the changes
- Merge the `develop` branch into the `master` branch
- You will get a merge conflict. Resolve the conflict by hand, using the file content of the `develop` branch. **Note:** You will require a text editor to merge `hello_c.py` by hand.
- Commit the changes. (Note in the commit message that you had to merge the file manually.)
- Push the changes of both branches to GitHub.

2.1 Git Resources

For most people, it takes some time to understand how Git works and get comfortable using it. Fortunately, there are very good resources online. You can just google and look for your preferred style of resource, and you will find loads of tutorials/videos/books etc. Just a few pointers to what we think are good resources:

- https://git-scm.com/doc – Here you will find Git's official documentation, including a handy cheat sheet and videos. You can also find the Pro Git book there, which is a very advanced and in-depth book on Git.
- https://learngitbranching.js.org/ – An interactive visual playground that helps you understand Git's basics.
- https://github.com/git-game/git-game A fun game where in each level, you have to use Git commands to manipulate a repository and find the next hint.