Documentation: File Backup

Max Weise

3. Juni 2021

Inhaltsverzeichnis

1 Introduction

1.1 Motivation for the project

I like to reset my Laptop every 6 - 7 Months, so I can get rid of any unused programs or files. This laptop is also connected to a NAS (Network Attatched Storage, basically a HDD which transfers data over the local network), which allows me to easily keep a backup at any times and over multiple devices, as long as they are connected to the local network.
Now to simplify the process, and because I have some folders, which I'm working on constantly and over multiple devices, I wanted to streamline the process of regularly copying them to said NAS.
At first glance, a simple copy script should be enough to get everything done automatically, but because I'm not the most organized person when it comes to my folders, I implemented the possibility to search for specified filetypes and only copy those to the NAS. That's the reason this project even exists. I'd like to share it with the programmer community, because either someone has the same problem as me, or is just curious to explore the programming world.

1.2 Disclaimer and Licence information

I won't take any responsibility for any damages to your system or loss of files, in case you use this program. I also won't guarantee to maintain the code in any way. This started as and will continue to be a side project which exists because I like to code and experiment with new stuff.
I don't mind the use, distribution or modification of this software. Please credit me or the Github repo in any way.
For more information, please read the **LICENCE** file in the repository.

2 Description of the Code

2.1 General Information

There are three main files in this project. All of those are located in the `cmd/src/` directory.
`file_backup` is the baseclass for this project, `file_type_backup` inherits from this class. This is a remain from the old version, where `file_type_backup` would sort out unwanted files and call the backup method of `file_backup`. I kept it that way, so I can reuse the constructor, getters and setters for the `root` and `dest` attributes.

2.2 Overview of Classes and Methods

`main.py` [1]

Run this file to start the program. To interact with it, input the number of the corresponding option.

`File_Backup`

parameters

- root - root directory; this will get copied
- dest - destination directory; this is the location the class copies to

methods
`__init__(self, root: str, destination: str)`
Initialize a File_Backup Object. All arguments are of type string and mandatory.

`set_root(self, new_root: str) -> None`
Set the *root* attribute to the argument *new_root*.

`set_dest(self, new_dest: str) -> None`
Set the *dest* attribute to the argument *new_dest*.

`backup_tree(self) -> None`
Copy the *root*-directory to the specified *dest*-directory. If *dset* does not exist, it will be created.
If the specified *root* is a file, a *ValueError* will be thrown.
If the specified *root* does not exist, a *ValueError* will be thrown.

`__str__(self) -> str`
Return a string representation of the object. This can be used to print to the terminal or in a logger.

`File_Type_Backup`

parameters

- root - root directory; this will get copied

[1] This module is currently just a rough draft, correct implementation will follow

- dest - destination directory; this is the location the class copies to
- __file_types - List of files that should be copied

Inherited attributes

- root
- dest

Inherited methods

- set_root(self, new_root: str)
- set_dest(self, new_dest: str)
- __str__(self)

methods

`__init__(self, root: str, dest: str, file_types: list)`
Initialize a *File_ Type_ Backup* object. Pass *root* and *dest* to the superconstructor.

`set__file_types(self, new_file_types: list) -> None`
Set the private attribute *__file_ types*

`get__file_types(self) -> list`
Return the list of file types currently set.

`__check_for_relevant_files(self, list_to_check: list) -> list`
Search for files, that should be backed up, according to the file types specified in *self.__file_ types*. Return a list containing all files that match the search criteria.

`backup_file_types(self) -> None`
Copy files from *root* to *dest*. Files get copied, if their extension (the file type, e.g.: „java“). If the *dest* directory does not exist, it will be created.
If the specified *root* is a file, a *ValueError* will be thrown.
If the specified *root* does not exist, a *ValueError* will be thrown.

3 Code Conventions

In this chapter you will find all conventions to ensure uniform code. In general, the *Flake8 linter* is used to enforce uniform code.
It can be installed using *pip*: `pip install flake8`. The corresponding config file is in the root directory.

3.1 General Overview

Starting at the top of the file, each module should have a docstring, containing information about

- What the module does
- When was the module created
- Date of last modification
- Author of the file (creator)

Docstrings start on line one with the usual three quotation marks. The description also begins on line one. If the description is longer than one line, the next line should be indented to match with the previous line.
Dates of creation and last modification as well as the author don't need to be indented. The following is an example of a docstring following the styleguide:

```
""" This is a description
    wich is longer than one line

Created 01.01.2021
Last Modified 21.03.2021
@author Max Weise
"""
```

If any modules have to be imported, the imports get organized like so:

1. Imports, which import whole modules (eg.: `import sys`)
2. Imports, which utilize the keyword `from` (eg.: `from tkinter import Tk`)
3. Imports, which import classes or functions defined in the project
   (eg.: `from file_backup import File_Backup`)
   Those modules will always be imported using the `from` keyword to avoid long lines.

<++>

3.2 Classes

3.3 Methods