BTM 2017 – Git practical

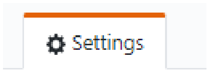
A bank decide to create a software to manage its clients. A team of developpers is hired for this job. They will use git and the plateform GitHub to host the project.

After speaking with the bank, they decide to split the work in sub-tasks that can be devolopped and tested independantly (still some module depend on others). Every time a module is developed, a branch is created (to not mess with the latest stable version). After adding a module and testing it, the branch can be merged to the master branch, fetched/merged with the remote master branch, and finally pushed to the GitHub repository.

I – Setting the Github and cloning the repository

Fork my repository on GitHub   (1 fork per group)
https://github.com/Tom-TBT/git_btm2017

Add collaborators to your copy



Clone the repository on each computer

II – Description of the modules to add

TO AVOID ANY CONFLICT IN MERGING, DON'T TOUCH ANYTHING ELSE THAN THE FUNCTION THAT YOU ARE SUPPOSED TO EDIT
EVEN A MISPLACED LINE SPACE CAN CREATE PROBLEMS

1)

functionalities.py   say_hi(): Display a nice message from the bank.
TEST: func.say_hi()

file_manager.py   add_client(client_name): Open "clients.txt" in writting mode and add the client specified by client_name. It don't have to check that the client already exist.
TEST: fM.add_client("boby")

file_manager.py   add_transaction(debtor, creditor, amount): Open "transaction.txt" in writting mode and add the transaction to the file in this way. For example is john is the debtor, bob the creditor and the amount is 2000:   john bob 2000
TEST: fM.add_transaction("john-rambo","marcel", 1520)

2)

file_manager.py get_clients(): Open "clients.txt" in reading mode, read all the names present in the files, put them in a list (without the line_return) and return the list.
TEST: fM.get_clients()

file_manager.py get_transactions(): Open "transactions.txt" in reading mode, read all the transaction present in the files, put them in a 2D list ([0]: debtor [1]: creditor [2]: amount) and return the list.
TEST: fM.get_transactions()

3)

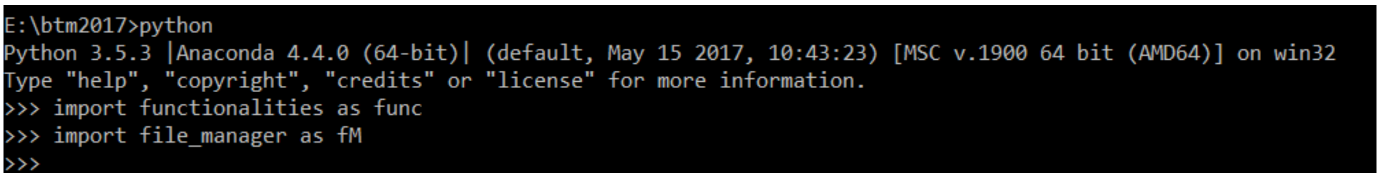
functionalities.py new_client(): Ask for the name of the new client, and if it don't already exist (use get_clients from file_manager.py), add it to the list via add_client in file_manager.py
TEST: func.new_client ()

functionalities.py new_transaction(): Print the list of the clients, ask for the debtor, the creditor and the amount. If the creditor and debtor exists, add the transaction via add_transaction in file_manager.py
TEST: func.new_transaction ()

functionalities.py look_credit(): Print the list of the clients, ask for the client to check on. Then from the list of transaction (use get_transactions from file_manager.py), calculate the money the client have (it can be negative amount).
TEST: func.look_credit()

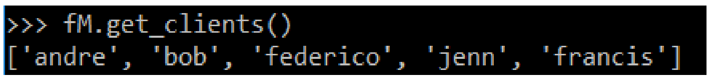
III – Testing a module

Test the modules with python. For that import functionalities and file_manager like bellow:



```
E:\btm2017>python
Python 3.5.3 |Anaconda 4.4.0 (64-bit)| (default, May 15 2017, 10:43:23) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import functionalities as func
>>> import file_manager as fM
>>>
```

Then use the TEST given for each module description:



```
>>> fM.get_clients()
['andre', 'bob', 'federico', 'jenn', 'francis']
```

III – Git workflow

For the development of each function, use the following git workflow. Do the command in that order (commands to look at git informations are repeated so you can see how git history evolve).

In green, the command that are used to check at the log/history/..., for you to know what's happening in git.
In blue, the steps that don't involve git.

1) Branching

git branch <name_of_the_branch>	#Create a branch <name_of_the_branch>
git branch	#Gives info about your branches
git checkout <name_of_the_branch>	#Moves to <name_of_the_branch>
git branch	#Gives info about your branches

2) Modifiing and commit

git status	#Gives info about files states

Do at this point modification according to specifications or copy paste the solution

git status	#Gives info about files states
git diff <file_you_edited>	#Show more details about modifications
git add <file_you_edited>	#Add <file_you_edited> to the stage
git status	#Gives info about files states
git log	#Show the modification history
git commit -m "Clear message"	#Add changes in the stage to the git history
git log	#Show the modification history
git status	#Gives info about files states

3) Merging and pushing work

git checkout master	#Moves to branch called master
git merge <name_of_the_branch>	#Merge the branch <name_of_the_branch> onto master
git remote	#Show info about the remote (connected to the server)
git fetch origin	#Collect modification history of the server
git merge origin/master	#Merge the branch origin/master onto master

If git shows conflicts here, ask for help

git log –graph –oneline	#Show the modification history (with a nice display)
git push origin	#Push your local modification into the server

Check onto the GitHub repository that your modification are there