nbgrader_toolbox

Python documentation

Name	Date modified	Type	Size
create_custom_config.py	4.9.2020 13.03	PY File	2 KB
get_grades_to_excel.py	4.9.2020 13.03	PY File	3 KB
git_assignments.py	4.9.2020 13.03	PY File	1 KB
late.py	4.9.2020 13.03	PY File	1 KB
moodle_submission_sort.py	4.9.2020 13.03	PY File	5 KB
old_nbgrader_config.py	4.9.2020 13.03	PY File	33 KB
print_assignments.py	4.9.2020 13.03	PY File	2 KB
release_assignment.py	4.9.2020 13.03	PY File	1 KB
send_feedback_to_students.py	4.9.2020 13.03	PY File	3 KB
synchronize_students.py	4.9.2020 13.03	PY File	2 KB
upload_grades.py	4.9.2020 13.03	PY File	2 KB

The configurator of the entire nbgrader application is basically the **nbgrader_config.py** file, which can be found in the root folder of the course. This file contains all the options, that are configured, so the application runs according to our needs.

create_custom_config.py

For the Machine Vision purposes, we want to specify certain constants, that ensure that our notebooks will get autograded properly. See on the image below, what the lines mean.

```
# THIS IS A MACHINE VISION CONFIG FILE

c = get_config()

#------------------------------------------------------------------------------
# AssignLatePenalties(NbGraderPreprocessor) configuration
#------------------------------------------------------------------------------

## Preprocessor for assigning penalties for late submissions to the database

## The plugin class for assigning the late penalty for each notebook.
# c.AssignLatePenalties.plugin_class = 'nbgrader.plugins.latesubmission.LateSubmissionPlugin'
c.AssignLatePenalties.plugin_class = 'late.SubMarks'

#------------------------------------------------------------------------------
# ClearSolutions(NbGraderPreprocessor) configuration
#------------------------------------------------------------------------------

## The delimiter marking the beginning of a solution
c.ClearSolutions.begin_solution_delimeter = '# ---------- YOUR CODE STARTS HERE -----------'

## The code snippet that will replace code solutions
c.ClearSolutions.code_stub = {'python': '# ---------- YOUR CODE STARTS HERE -----------\n\n\n\n\n# ----------- YOUR CODE ENDS HERE ------------',
                              'matlab': "% YOUR CODE HERE\nerror('No Answer Given!')",
                              'octave': "% YOUR CODE HERE\nerror('No Answer Given!')",
                              'java': '// YOUR CODE HERE'}

## The delimiter marking the end of a solution
c.ClearSolutions.end_solution_delimeter = '# ----------- YOUR CODE ENDS HERE ------------'

## The text snippet that will replace written solutions
c.ClearSolutions.text_stub = ``REPLACE THIS TEXT WITH YOUR ANSWER.``
```

Late penalties are assigned through the late.py script

This Machine Vision config file gets created when we create the course folder and it is done so by calling the **create_custom_config.py**. If you would like to change, what will be written into the custom config file, checkout the **old_nbgrader_config.py**, which contains all the possible configurations that nbgrader accepts. If some of those options seems like a usable option, copy and paste it into the create_custom_config.py into the multiline text field. Then create new course folder.

The config files are based on the traitlets configurable object. It is a very useful tool how to provide the end user a high level of customizability of the application. If you want to learn more about this, checkout the documentation, especially this link: https://traitlets.readthedocs.io/en/stable/config.html

Late.py

This file contains the late submission policy. The function **late_submission_penalty** returns, how many points are supposed subtracted from the score.

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 15 13:18:30 2020

@author: duher18
"""


from __future__ import division
from nbgrader.plugins import BasePlugin


class SubMarks(BasePlugin):
    def late_submission_penalty(self, student_id, score, total_seconds_late):
        """Penalty of 1 mark per hour late"""
        # hours_late = total_seconds_late / 3600
        # return round(hours_late, 0)

        """Machine Vision penalty policy"""
        if total_seconds_late == 0:
            return score
        elif total_seconds_late > 0 and total_seconds_late <= 172800: # 0-2 days
            return score*0.25
        elif total_seconds_late > 172800 and total_seconds_late <= 345600: # 2-4 days
            return score*0.5
        elif total_seconds_late > 345600 and total_seconds_late <= 518400: # 4-6 days
            return score*0.75
        else:
            return score*1
```

If you would like to add another policy, please feel free to do so! Best way would be commenting the other policies and writing your own and then it would be nice if the updated late.py file would be pushed into the git repository. That way, we can collect all the different policies and all there is to do is to uncomment the one we want.

Remember no other tweaking is needed, simply provide the late submission policy here. The only thing is that only the score and total_seconds_late are available. However, that should be all the information that is needed for majority of late policies.

get_grades_to_excel.py

This is an older function, which is not in use anymore, however it is still left in here, if someone would like to generate an excel file with the grades.

git_assignments.py

This script pulls the master assignments from the private repository course_assignments.git. The idea was to create this repository to be storing the master versions of the assignments and anybody would be able to immediately pull them. Checkout the official nbgrader documentation on how to create the master versions in the first place:
https://nbgrader.readthedocs.io/en/stable/user_guide/creating_and_grading_assignments.html

moodle_submission_sort.py

By far the most complicated script of them all. The Moodle submissions are downloaded as a zip file and moved into the submitted folder. Then, this script is responsible for sorting the submissions into the hierarchy that nbgrader understands.

According to my testing, it is working properly now, however I am sure that it will require more tweaking when the real student submissions start coming in and some unexptected errors for example due to naming might appear.

print_assignments.py

prints a structured table of the available assignments from the source folder and shows, which one of them have already been released. Or, in other words, for which master versions have the student versions already been generated.

release_assignment.py

Very important file, because not only it creates the student version of the assignment, but it also updates the due date, which is crucial for correct assessment of the late submissions.

synchronize_students.py

Updates the gradebook.db file according to the downloaded participants.csv file from Moodle. Due to some errors, at this point, the gradebook will get renewed when this script gets called and some previous data about the students is removed from the gradebook.db. Because of that, make sure that all the grades are uploaded to Moodle after the grading is finished and don’t rely on storing important information within the gradebook.db file.

upload_grades.py

generates a csv file containing the grades. This csv file can be uploaded to the Moodle course page. Please follow the instructions in the github page on how to import the csv file into the Moodle course.

send_feedback_to_students.py

The script logs into your email and cycles through all the available feedbacks and sends them to the corresponding authors.

!! IMPORTANT !! Within the **send_feedback_to_students.py**, find the section, where the variable body gets declared and change the body of the email, change your signature, etc.

It is possible to play with the structure of the body, add HTML, etc. Checkout this link for further inspiration

https://docs.python.org/3.4/library/email-examples.html

```
feedback_folder = os.getcwd()+ '\\feedback

with Gradebook('sqlite:///gradebook.db') as gb:

    feedback_students = os.listdir(feedback_folder)

    for student_id in feedback_students:

        # --- setup the message ---
        msg = MIMEMultipart()
        # assignment_name = 'A1-Imaging'

        msg['Subject'] = 'MV_'+assignment_name.split('-')[0]+' Feedback'

        msg['From'] = my_address

        body = """
        Hi!

        Please find the feedback for the assignment {} attached.

        Best regards,
        Teacher


        """.format(assignment_name)
        student = gb.find_student(student_id)
        msg['To'] = student.email
        student_feedback = feedback_folder+'\\'+str(student.id)+'\\'+assignment_name
```