Otter-Grader Demo Notebook

This notebook demonstrates how to use otter-grader. For installation details, see the README (https://github.com/ucbds-infra/otter-grader).

Student Usage

Notebooks

Otter supports in-notebook checks so that students can check their progress when working through an assignments. To use it, you need to create an instance of `otter.Notebook`, to which you (optionally) pass a path to a directory of tests; this defaults to `./tests`.

To check a student's work, use `Notebook.check`. Pass to this the question identifier, which is the filename (without the `.py` extension). For example, in this demo, `./tests/q4.py` tests a student's `square` function:

If we have the square function defined below, we can run the test using `grader.check("q4")`:

If we change the function so that the tests fail, then the grader outputs the failed test and the incorrect output.

Students can also run all tests at once using `Notebook.check_all`:

Students can also create their own PDF submissions using `Notebook.export`, to which you pass the path to the notebook. Otter uses nb2pdf to generate PDFs (more info below). The filtering behavior of nb2pdf is turned on by default, although it can be turned off using the `filtering=False` argument.

Scripts

If a student is working with Python scripts instead of notebooks, the otter command line utility has a `check` command that can run these tests as well. The help entry for this command is given below.

If we wanted to run a single check, we would pass the question identifier (the filename without `.py`) to the `-q` flag. The tests path is assumed to be `./tests` unless another path is provided to the `-t` flag.

As with notebooks, a failed test will output the test and the incorrect output.

To check all of the tests at once, use `otter check` without a `-q` flag:

Grading

Notebooks

Otter uses Docker containers to execute students' submissions in parallel containers. It accepts exports from Gradescope and Canvas, or the notebook files with a JSON- or YAML-formatted metadata file. If you're using the custom metadata file, it requires an entry for each notebook, with the notebook filename as the `filename` parameter and the student identifier as `identifier`. An example of the YAML metadata can be found in `notebooks/meta.yml`.

Requirements

The docker container comes with the following packages installed:

- datascience
- jupyter_client
- ipykernel
- matplotlib
- pandas
- ipywidgets
- scipy
- nb2pdf
- otter-grader

If you have any other requirements besides these, create a `requirements.txt` file:

Command-Line Usage

Now let's examine the usage of the `otter` command:

The `-p` flag should be the path to the directory that conains the notebooks (or the unzipped export from Canvas or Gradescope). The `g`, `c`, `y`, and `j` flags indicate the metadata type; the `y` and `j` flags require an argument that indicates the path to the metadata file. If you want to generate PDFs for manual grading, you can add the `--pdf` or `--filter-pdf` flags, which generate an unfiltered or filtered, respectively, PDF for each notebook. If you need a requirements file, pass that path to the `-r` flag. Optionally, you can set the number of docker containers or the docker image that the grader runs on using the `--containers` and `--image` flags, respectively.

Some flag defaults are listed below. If a flag is not list, it has no default value.

Flag	Default
`-p`	`./`
`-t`	`./tests`
`-o`	`./`
`--containers`	4
`--image`	`ucbdsinfra/otter-grader`

If your notebooks require any other files (e.g. data files), place these files into the notebooks path directory (whatever the value of `-p` is). These will automatically be copied into the docker containers.

Let's run the autograder on the files in `./notebooks`. The command below tells otter that our metadata file can be found at `./notebooks/meta.yml`, that the notebooks can be found in `./notebooks`, that we have a requirements file at `./requirements.txt`, and to use verbose output.

We should now have a CSV file with the scores and score breakdowns at `./final_grades.csv` (since we left `-o` with its default value).

If the autograder ran correctly, all of the submissions should receive a 2/5, since three of the tests are written to fail on these notebooks.

nb2pdf

When exporting notebook PDFs, otter uses the library nb2pdf, which in turn relies on nbpdfexporter. This requires that chromium be installed on the docker container (which is true if using `ucbdsinfra/otter-grader`, the default image) but it also must be installed on the JupyterHub distribution that students use if they are going to be exporting PDFs themselves.

When generating the PDF (if filtering is turned on, either by use of the `--filter-pdf` flag or `filtering=True` in `Notebook.export`), the following cells will be included by default:

- Markdown cells
- Code cells with images in the output
- All cells tagged with `include`

If a cell falls within one of the 3 categories listed above but you do not want to include it in the exported PDF, tag the cell with `ignore`. This cell will then not be added to the manual graded PDF.

Scripts

Otter can also grade Python scripts using the addition of the `-s` flag. It uses the same metadata and path structure as with notebooks, so the only change to our call above will be to change the path to `./scripts`. *This will overwrite `final_grades.csv` from above since we aren't changing the output path.*

Now, we should be able to read in the grades from the scripts. If all went well, they should each have a 4/5.

Gradescope

Otter is compatible with the Gradescope autograder, and has a command line tool to generate the zipfile that is used to configure the Docker container that Gradescope's autograder usage. The base command for this utility is `otter gen`, and its help entry is given below.

The `otter gen` command creates a zipfile at `OUTPUT-PATH/autograder.zip` which contains the necessary files for the Gradescope autograder:

- `run_autograder` : the script that executes otter
- `setup.sh` : a file that instructs Ubuntu on how to install dependencies
- `requirements.txt` : Python's list of necessary dependencies
- `tests` : the folder of test cases
- `files` : a folder containing any files needed for the notebooks to execute (e.g. data files)

The requirements file create automatically includes the otter dependencies (see Requirements), but you can optionally include your own, other ones by passing a filepath to the `-r` flag. Any files included that are not passed to a flag are automatically placed into the `files` folder in the zipfile and will be copied into the notebook directory in the Gradescope container.

Let's generate a zipfile from this directory with the tests in `./tests` , the requirements in `./requirements.txt` . and `test-df.csv` as a data file.

Now, we have a file `./autograder.zip` in the working directory:

If we unzip the file, you'll see all of the files listed above as well as the CSV file we added in the `autograder/files` directory:

Pass/Fail Thresholds

The generator also supports scoring assignments on a pass/fail basis using only some percentage threshold. If you include a threshold, pass a number between 0 and 1 to the `--threshold` flag. If a student scores below `--threshold` , then they will get 0 points; otherwise, they will get points for all questions.

The command below creates an autograder that automatically gives students full points as long as they get at least an 80%.

Overriding Points Possible

Otter also supports overriding the number of points possible for an assignment from the default to the sum of the points possible for each test using the `--points` flag. This is done by scaling the new points possible by the percentage of points received as a decimal. For example, if a student passes a 2- and 1-point test but fails a 4-point test on an assignment with `--points` set to 2, they will receive a $\frac{2+1}{2+1+4} \cdot 2 = 0.85714285714$.

The command below creates an autograder that scales the number of points down to 3.

Relative Imports on Gradescope

Because of the way that the Gradescope autograder is set up, our script must change relative import syntax in order to ensure that the necessary files are loaded. **Because we cannot differentiate between package imports and relative imports, `otter gen` automatically assumes that if you are importing a .py file that it is called `utils.py`.** The regex used to change the import syntax will *only* change the syntax if you're using the import statment

```
from utils import *
```

For this reason, if you have any functions you need in a file that is going to be imported, *make sure that it is called `utils.py`.*