cornellGrading

Release 2.16.2

Dmitry Savransky

Mar 25, 2024

CONTENTS:

CHAPTER

ONE

SETUP

cornellGrading setup involves two steps:

1. Installation
2. API Token setup

Both steps must be completed prior to using the code.

1.1 Installation

1.1.1 From PyPI (recommended)

To install from PyPI:

```
pip install --user cornellGrading
```

Or, with optional dependencies required to push LaTeX into Canvas HTML:

```
pip install --user cornellGrading[latex2html]
```

To install system-wide, omit the `--user` option. This requires administrative privileges on most systems.

Note: The `latex2html` option requires the pandoc executable to be installed and in the system PATH. For detailed pandoc installation instructions see here: https://pandoc.org/installing.html

---

1.1.2 From GitHub

If cloning from github, in the cloned grading directory:

```
pip install --user .
```

or, to install in developer mode:

```
pip install --user -e .
```

In order to also install requirements needed push LaTeX into Canvas HTML, do:

```
pip install --user -e .[latex2html]
```

Note: To upgrade to the latest version, just append `--upgrade` to whichever install command you originally used. For example: `pip install --upgrade --user cornellGrading`.

1.2 Canvas API Token (Required)

To generate a token, in Canvas:

1. Navigate to Account>Settings and scroll down to Approved Integrations
2. Click '+New Access Token'. Copy the token. **NB: It won't be displayed again.**

You will need to enter this token the first time you instantiate a *cornellGrading.cornellGrading* object. If using Windows, you should save this token to a text file. Be sure that there is nothing other than the token in the file (white space afterwards is ok).

Warning: Windows users working with a standard shell will be unable to copy/paste this token into the command prompt. Windows users should use the `canvas_token_file` input when instantiating *cornellGrading* for the first time, or retype it into the prompt. Using a text file is recommended.

Note: The token is stored in your system's keychain as `canvas_test_token1` and will be automatically loaded on all subsequent *cornellGrading.cornellGrading* instantiations. If you need to change the local token, you must manually delete it from the keychain. The token is entered as a secure password, and so you will not see the cursor move as you enter it. The token will only be saved if the connection to Canvas is successful.

In order to load the token into your system's keychain, in python:

```
from cornellGrading import cornellGrading
c = cornellGrading(canvas_token_file=r'path_to_token_file') #replace with fullpath to␣
↪the text file with your token
```

You can also omit the `canvas_token_file` input, in which case you will be prompted to enter your token at the command line.

Note: If entering your token at the prompt, you will not see the cursor move. Just hit Enter when done. Windows users must type the token out. Others can copy/paste.

1.3 Qualtrics API Token (Optional)

On the qualtrics site:

1. Navigate to: Account Settings>Qualtrics IDs
2. Click the 'Generate Token' button under API
3. This page also lists all other IDs you need to know

You will need to enter this token the first time you run *setupQualtrics()*. If using Windows, you should save this token to a text file. Be sure that there is nothing other than the token in the file (white space afterwards is ok).

Warning: Windows users working with a standard shell will be unable to copy/paste this token into the command prompt. Windows users should use the `qualtrics_token_file` input when running *`setupQualtrics()`* for the first time, or just retype it into the prompt. Using a text file is recommended.

Note: The token is stored in your system's keychain as `qualtrics_token` and will be automatically loaded on all subsequent *`setupQualtrics()`* calls. If you need to change the local token, you must manually delete it from the keychain. The token is entered as a secure password, and so you will not see the cursor move as you enter it. The token will only be saved if the connection to Canvas is successful.

In order to load the token into your system's keychain, in python:

```
from cornellGrading import cornellQualtrics
c = cornellQualtrics(qualtrics_token_file=r'path_to_token_file') #replace with fullpath␣
↪to the text file with your token
```

You can also omit the `qualtrics_token_file` input, in which case you will be prompted to enter your token at the command line.

Note: If entering your token at the prompt, you will not see the cursor move. Just hit Enter when done. Windows users must type the token out. Others can copy/paste.

1.4 Qualtrics De-Anonymization

By default, Cornell anonymizes all survey responses, regardless of how you have set up your survey. To fix this, email itservicedesk@cornell.edu and request that they toggle "View Restricted Data" to On for your qualtrics account.

CHAPTER

TWO

COURSE WORKFLOWS

2.1 Initial Course Setup

In Canvas:

1. Navigate to Settings>Course Details and change course name to something unique (suggestion: Name: Semester). Be sure to click 'Update Course Details' at the bottom.
2. Navigate to Settings>Course Details, scroll to the very bottom, click 'more options', and check 'Hide totals in student grades summary'. Click 'Updated Course Details'
3. Navigate to Settings>Navigation and ensure that 'Grades' and 'Assignments' are both in the active items (top part of page). If you want students to be able to directly access files (rather than only via links), then add 'Files' to the active navigation items as well. Don't forget to click 'Save' at the bottom of the page.
4. Go to Student View (button on right-hand side of home page), exit student view, and then ensure that the 'test student' appears in People when you are back in Instructor view (only necessary if you want the test student to be part of the qualtrics mailing list for debugging purposes).
5. Navigate to Grades. Click the settings icon (right of the search box) and go to the 'Grade Posting Policy tab'. Ensure that 'Automatically Post Grades' is clicked (this allows for students to see comments on HWs before grades are entered, which is necessary for link injection to the self-grading surveys). Be sure to click 'Update' if any changes are made.

Now, in python (this assumes that you have completed all steps in *Setup*):

```
from cornellGrading import cornellGrading
c = cornellGrading()

#get your coursenumber (the part in parentheses):
print("\n".join(c.listCourses()[0]))

coursenum = ... #change to your actual number from list printed above

c.getCourse(coursenum)

#consistency check
print(c.coursename) #should be the course name you set in Canvas
print(c.names) #should be all your students

#connect to qualtrics and generate course mailing list
#(skip if you don't care about qualtrics)
c.setupQualtrics()
c.genCourseMailingList()
```

2.2 Upload a Homework and Create a New Assignment

This procedure automates the creation of new assignments, assuming that your homework statement is in a single PDF document. This assumes that there already exists an 'Assignments' assignment group (Canvas default), and will create a 'Homeworks' folder under Files that is not directly accessible to students (assuming it does not already exist).

The assignment will be titled 'HW?' where ? is the assignment number (i.e., 'HW1', 'HW2', etc.).

In python:

```
from cornellGrading import cornellGrading
c = cornellGrading()
coursenum =   #insert your course number here
c.getCourse(coursenum)

assignmentNum =   #enter assignment number (must be integer)
duedate =         #enter assignment duedate in 'YYYY-MM-DD' format. The due time will be␣
↪5pm by default.
hwfile =          #string - full path on your local disk to the HW pdf file
res = c.uploadHW(assignmentNum,duedate,hwfile)

#by default, the created assignment will be worth 10 points.  To change this, instead␣
↪run:
res = c.uploadHW(assignmentNum,duedate,hwfile,totscore=N) #N must be an integer

#by default, the created assignment will be immediate visible. To change this, instead␣
↪run:
res = c.uploadHW(assignmentNum,duedate,hwfile,unlockDelta=M)
#where M is a positive float and represents the number of days prior to the due date to␣
↪unlock the assignment.
```

2.2.1 Injecting Homework Text into the Canvas Assignment

If the homework file is a PDF compiled from LaTeX source code, and resides in the same directory as the original `tex` file, then you can inject its contents directly into the homework assignment as HTML, along with the link to the PDF, by running:

```
res = c.uploadHW(assignmentNum, duedate, hwfile, injectText=True)
```

See *LaTeX to Canvas* for further details.

Note: `hwfile` must point at the PDF in the directory where it was compiled, and all other required files (figures, etc.) must reside in this same path.

2.3 Create a HW Survey

This assumes that you have set up your assignment with the name 'HW?' where ? is the assignment number (i.e., 'HW1', 'HW2', etc.).

```
from cornellGrading import cornellGrading
c = cornellGrading()
coursenum =   #insert your course number here
c.getCourse(coursenum)
c.setupQualtrics()
assignmentNum = 1 #change to actual assignment number
nprobs = 3 #number of problems in assignment
c.setupPrivateHW(assignmentNum,nprobs)
```

Or, let's say you're a weirdo who only wants a single grade for the whole assignment, and wants the students to grade themselves out of 10,9,7,5,3, exclusively. Then the last line becomes:

```
c.setupPrivateHW(assignmentNum,0,scoreOptions=[10,9,7,5,3])
```

After executing (assuming no errors), you should see a new survey in Qualtrics with the name "Coursename HW? Self-Grade", and a personalized link should be injected into the comments for each student in the original assignment.

If your course roster has changed, be sure to run `c.updateCourseMailingList()` prior to `setupPrivateHW`.

You can also share the created survey with another qualtrics user (say, your TA). To do so, you will need them to give you their Qualtrics id, which they can find in the Qualtrics IDs page ([see Qualtrics API Token](#qualtrics-api-token)). Make sure you get their ID, and not their API token. To enable sharing, add `sharewith=qualtricsid` to the `setupPrivateHW` call, where `qualtricsid` is id string to share with.

2.4 Upload Solutions and Create Self-Grading Assignment

In addition to creating the HW survey in qualtrics and injecting links into the assignment comments, `setupPrivateHW` can also create a self-grading assignment on Canvas with the homework solutions and a due date that is different from the due date of the original assignment. This functionality is toggled by passing `createAss=True` to the `setupPrivateHW` call. The other relevant keyword arguments are:

- `solutions`: String, full path to solutions PDF file on your local disk
- `selfGradeDueDelta`: Float, number of days after original assignment due date for self-grading to be due (defaults to 7)
- `selfGradeReleasedDelta`: Float, number of days after original assignment due date when the self-grading assignment is released to students (defaults to 3).

So, a full call would look something like:

```
from cornellGrading import cornellGrading
c = cornellGrading()
coursenum =   #insert your course number here
c.getCourse(coursenum)
c.setupQualtrics()
assignmentNum = 1 #change to actual assignment number
nprobs = 3 #number of problems in assignment
solutionsFile =   #insert path to solutions file
c.setupPrivateHW(assignmentNum,nprobs,createAss=True,solutions=solutionsFile)
```

This will create a 'Homework Self-Grading' assignment group (if it does not already exist), and will create a 'Homeworks' folder under Files that is not directly accessible to students (also assuming it does not already exist).

2.5 Grab Self-Grading Results and Upload to Canvas

Finally, once students have completed their self-assessment via Qualtrics, we need to move their scores into the Canvas gradebook. This is done via the *selfGradingImport()* method. Again, this assumes that you have set up your assignment with the name 'HW?' where ? is the assignment number, and also that you have assigned a point value to the assignment in Canvas (if you're using the single-question survey variant, and not checking for late submissions, the latter is not required).

In python:

```
from cornellGrading import cornellGrading
c = cornellGrading()
coursenum =   #insert your course number here
c.getCourse(coursenum)
c.setupQualtrics()
assignmentNum = 1 #change to actual assignment number
c.selfGradingImport(assignmentNum)
```

By default, this will take the sum of all of the survey question responses, scale by the ratio of the total assignment points (grabbed from Canvas) to the total number of possible points in the survey. If you are using the single-question survey variant (i.e., set `nprobs` to 0 in the `setupPrivateHW` call), then the assignment total value in Canvas is ignored, and just the exact value from Qualtrics is used.

Default behavior is to check for late submissions, and then subtract 1/4th the total number of points if the assignment is late. Lateness is defined by the `maxDaysLate` keyword (defaults to 3), past which the assignment is marked zero, and the penalty itself is set by `latePenalty`. In order to toggle off late checking alltogether, set `checkLate=False`, so that the last line above becomes `c.selfGradingImport(assignmentNum,checkLate=False)`.

If your assignment has extra credit problems, you can identify these in your survey by adding the words 'Extra Credit' to any of the question names. In this case, a maximum of `ecscore` points (default is 3) will be added to the HW score for all extra credit problems being self-marked perfect (and scaling down consistently with self-grading).

CHAPTER

THREE

LATEX TO CANVAS

Note: This functionality requires that the package has been installed with the optional `[latex2html]` dependencies, and that you have pandoc installed and in the system PATH. See *Setup* for further details.

The *latex2html()* method allows you to convert LaTeX source into Canvas-native html, including rendering equations via the Canvas equation editor (this produces an equation image that contains the original equation as alt tex and (in compatible browsers) a MathJax payload that can be utilized by screen readers. This is the single best approach (found to date) of ensuring that LaTeX-derived products are fully accessible.

This functionality uses pandoc for the initial HTML conversion, and is therefore limited by pandoc's capabilities. For details on pandoc, see the user's guide: https://pandoc.org/MANUAL.html

In general, pandoc will be able to handle relatively simple documents based on the article or amsart class, using some additional optional packages (see: https://pandoc.org/MANUAL.html#variables-for-latex). Macros defined in the source will typically be supported (unless they rely on unsupported packages). User-defined style files typically will not work. The easiest way to see whether your LaTeX document will render properly is to run pandoc on it manually and inspect the output html.

The *latex2html()* method will attempt to do the following:

1. Make a copy of the input LaTeX source and apply a dictionary of standard substitutions. The dictionary variable name is `texsubdict`. It currently only contains mappings from `\nicefrac` to `\frac` and removes all `\ensuremath` directives, but can be updated as needed.
2. Execute pandoc on the cleaned-up source code, using the original direcotry as the working directory, with flags `--webtex` and `--default-image-extension=png`
3. Read in the resulting HTML and parse. In parsing, every equation image link will be overwritten to point at Canvas, every image (inside a Figure) will be uploaded to Canvas (if the original image is no a PNG, a PNG will be generated first) and the image link will be updated to point at the uploaded figure, and all span css formatting will be added to each individual span directive.

Currently, images will only be properly handled if inside a Figure float. Table support exists, but is spotty.

3.1 Create Canvas Page from LaTeX

The *latex2page()* method allows you to convert LaTeX source into a Canvas page.

Assuming you have instantiated a `cornellGrading` object as `c`, as above, you can run:

```
res = c.latex2page(fname, title)
```

where *fname* is the full path to either the LaTeX source or the PDF compiled from the source (which must be in the same directory as the source), and *title* is the title for the generated page. Other method options include:

- `insertPDF=True` will also include a link to the compiled PDF in the generated page (in this case *fname* must be the compiled PDF)
- `published=True` will automatically publish the page (the page is unpublished by default).

CHAPTER

FOUR

QUALTRICS WORKFLOWS

4.1 Quotas for Dynamic Response Updates

Use case: you wish to create a survey with a selection of one of many items and a standard set of questions related to that item (for example, ranking candidates). To avoid user error, you would like to remove items from the selection set once a response has been registered for them. This is relatively easy to do by setting up quotas for each item, and associating the quotas back to the display options in the item. Below is a worked example.

1. Create sample data:

```
items = ['Choice 1', 'Choice 2', 'Choice 3', 'Choice 4', 'Choice 5']
```

2. Set up qualtrics and create a survey

```
from cornellGrading import cornellQualtrics
c = cornellQualtrics()
surveyId = c.createSurvey('Test Quota Survey')
```

3. Add dropdown menu question for choices

```
desc = "Select Item"
choices = {}
for j, choice in enumerate(items):
    choices[str(j + 1)] = {"Display": choice}
choiceOrder = list(range(1, len(choices) + 1))
questionDef = {
    'QuestionText': desc,
    'DefaultChoices': False,
    'DataExportTag': 'Q1',
    'QuestionType': 'MC',
    'Selector': 'DL',
    'Configuration': {'QuestionDescriptionOption': 'UseText'},
    'QuestionDescription': desc,
    'Choices': choices,
    'ChoiceOrder': choiceOrder,
    'Validation': {
        'Settings': {
            'ForceResponse': 'ON',
            'ForceResponseType': 'ON',
            'Type': 'None'
        }
```

(continues on next page)

(continued from previous page)

```
    },
    'Language': [],
    'QuestionID': 'QID1',
    'QuestionText_Unsafe': desc}
qid1 = c.addSurveyQuestion(surveyId, questionDef)
```

4. Now we add some general multiple choice questions related to each item

```
nrubrics = 5
scoreOptions = [0, 1, 2, 3]
choices = {}
for j, choice in enumerate(scoreOptions):
    choices[str(j + 1)] = {"Display": str(choice)}
choiceOrder = list(range(1, len(choices) + 1))

for j in range(1, nrubrics + 1):
    desc = "Rubric %d Score" % j
    questionDef = {
        "QuestionText": desc,
        "DataExportTag": "Q%d" % (j + 1),
        "QuestionType": "MC",
        "Selector": "SAVR",
        "SubSelector": "TX",
        "Configuration": {"QuestionDescriptionOption": "UseText"},
        "QuestionDescription": desc,
        "Choices": choices,
        "ChoiceOrder": choiceOrder,
        "Validation": {
            "Settings": {
                "ForceResponse": "ON",
                "ForceResponseType": "ON",
                "Type": "None",
            }
        },
        "Language": [],
        "QuestionID": "QID%d" % (j + 3),
        "DataVisibility": {"Private": False, "Hidden": False},
        "QuestionText_Unsafe": desc,
    }
    c.addSurveyQuestion(surveyId, questionDef)
```

5. Now that we have the basic survey set up, we can add quotas for each item. There are no quota groups in the survey yet, so first you need to create a group, and then all quotas will be assigned to it by default.

```
quotaGroupName = "q1quotas"
quotaGroupId = c.addSurveyQuotaGroup(surveyId, quotaGroupName)

quotas = []
for j,s in enumerate(items):
    quotaDef = {
        'Name': 'name{}quota'.format(j+1),
        'Occurrences': 1,
```

(continues on next page)

(continued from previous page)

```
        'Logic': {'0': {'0': {'LogicType': 'Question',
                              'QuestionID': 'QID1',
                              'QuestionIsInLoop': 'no',
                              'ChoiceLocator': 'q://QID1/SelectableChoice/{}'.
↪format(j+1),
                              'Operator': 'Selected',
                              'QuestionIDFromLocator': 'QID1',
                              'LeftOperand': 'q://QID1/SelectableChoice/{}'.
↪format(j+1),
                              'Type': 'Expression',
                              'Description': ''},
                              'Type': 'If'},
                         'Type': 'BooleanExpression'},
        'LogicType': 'Simple',
        'QuotaAction': 'ForBranching',
        'ActionInfo': {'0': {'0': {'ActionType': 'ForBranching',
                                   'Type': 'Expression',
                                   'LogicType': 'QuotaAction'},
                                   'Type': 'If'},
                              'Type': 'BooleanExpression'},
        'QuotaRealm': 'Survey',
        'Count': 0}
    quotas.append(c.addSurveyQuota(surveyId, quotaDef))
```

6. As a last step, we need to redo the original first question to add display logic to each of the entries, associated with each quota

```
desc = "Select Item"
choices = {}
for j, choice in enumerate(items):
    choices[str(j + 1)] = {'Display': choice,
                           'DisplayLogic': {'0': {'0': {'LogicType': 'Quota
↪',
                            'QuotaID': quotas[j],
                            'QuotaType': 'Simple',
                            'Operator': 'QuotaNotMet',
                            'LeftOperand': 'qo://{}/QuotaNotMet'.
↪format(quotas[j]),
                            'QuotaName': 'name{}quota'.format(j+1),
                            'Type': 'Expression',
                            'Description': ''},
                            'Type': 'If'},
                            'Type': 'BooleanExpression',
                            'inPage': False}}
choiceOrder = list(range(1, len(choices) + 1))
questionDef = {
    'QuestionText': desc,
    'DefaultChoices': False,
    'DataExportTag': 'Q1',
    'QuestionType': 'MC',
    'Selector': 'DL',
    'Configuration': {'QuestionDescriptionOption': 'UseText'},
```

(continues on next page)

(continued from previous page)

```
    'QuestionDescription': desc,
    'Choices': choices,
    'ChoiceOrder': choiceOrder,
    'Validation': {
        'Settings': {
            'ForceResponse': 'ON',
            'ForceResponseType': 'ON',
            'Type': 'None'
        }
    },
    'Language': [],
    'QuestionID': 'QID1',
    'QuestionText_Unsafe': desc}

c.updateSurveyQuestion(surveyId, qid1, questionDef)
```

You will now have a survey where, after a submission is made for any item, the item will no longer appear in question 1 selection options upon reload of the survey.

CHAPTER

FIVE

EXTENDED EXAMPLES

5.1 Individualized Quizzes

I decided (for reasons that defy explanation), to give a personalized, self-administered, self-timed, written exam (hello to the other 3 people in the universe with this use case). Canvas, in their infinite wisdom, provide the exact framework for this kind of thing (in the form of graded Quizzes), but then do not allow a file upload to the quiz. So, we go the convoluted route: use a Quiz to deliver the exam to students and record a start time, and then use an assignment to collect the responses. Compare quiz start times to assignment submit times to check whether the students exceeded their allowed time window, and then grade the submissions as usual. All that remains is setting up a separate quiz for each student and assigning each one to *only* the intended student. Here we go:

1. First we set up a single assignment to collect the results. This can be done manually via the web interface or through the API, but in either case, note the assignment id and url so you can link directly to it from the quizzes (not strictly necessary, but makes things easier for the students).

2. Write your bank of exam questions. I decided that I wanted groups of similar questions, wanted to give each student 3 questions to solve, and had ~45 students, so I ended up with 3 groups of questions of 4, 4, and 3 questions in each (48 unique combinations). Each question lives in its own file named `prob1.tex`, `prob2.tex`, etc., and there's a `header.tex` in the same directory with all the front matter of the exam (including a `\begin{enumerate}` directive). We're going to generate and compile a unique pdf tagged with each student's netid - here I'll just show one instance, but you wrap everything in a loop to cover all students.

```
from cornellGrading import cornellGrading
from datetime import datetime, timedelta
import numpy as np
import subprocess

c = cornellGrading()
coursenum = ... # your course number here
c.getCourse(coursenum)

prelimpath = ... # path to prelim dir with all the tex files

# question groups
block1 = [1,2,3,4]
block2 = [5,6,7,8]
block3 = [9,10,11]

# generate unique combinations and randomize order
combos = []
for i in range(len(block1)):
```

(continues on next page)

(continued from previous page)

```
    for j in range(len(block2)):
        for k in range(len(block3)):
            combos.append([block1[i],block2[j],block3[k]])

combos = np.array(combos)
rng = np.random.default_rng()
arr = np.arange(len(combos))
rng.shuffle(arr)
combos = combos[arr]

# create/get destination folder on Canvas for uploads
prelimfolder = c.createFolder("Homeworks/Prelim", hidden=True)

#going to do just the first student, but this is where you'd start the loop
nid = c.netids[0] # netid
comb = combos[0]  # problem set

# write exam tex file
fname = os.path.join(prelimpath, 'prelim_2020_{}.tex'.format(nid))
with open(fname, "w") as f:
    f.write("\\input{header.tex}\n")
    for p in comb:
        f.write("\\input{prob%d.tex}\n"%(p))
    f.write("\\end{enumerate}\n")
    f.write("\\bigskip\\bigskip This exam is for %s"%(nid))
    f.write("\\end{document}\n")

# compile exam
_ = subprocess.run(["latexmk","-pdf",fname],cwd=prelimpath,check=True,
↪capture_output=True)
pdfname = os.path.join(prelimpath, 'prelim_2020_{}.pdf'.format(nid))
assert os.path.exists(pdfname)

prelimupload = prelimfolder.upload(pdfname)
assert prelimupload[0], "Prelim Upload failed."
```

Note that I've used `latexmk` (which of course has to be in my path already) to take care of multiple compilation steps, etc. If all your questions are simple (no cross-references or anything else requiring multiple compilations, then regular `pdflatex` should work fine instead).

3. At this point, we've generated a unique exam PDF for our student(s), so now it's time to place it into a timed quiz. We have to make it a graded quiz so it can show up in an assignment group. Another important caveat here is how Canvas handles question additions to existing quizzes. If the quiz is already published and you add a question, Canvas requires that you re-save the quiz before the question becomes visible to students. I have not found a good way of doing this via the API (what the hell, Canvas?), so the best solution is to generate the quiz *unpublished* add all your questions, and then edit the existing quiz object to make it published. As a final step, we need to add an assignment override onto the quiz assignment to give it a due date, an unlock date, and to assign it to a single student. Again, showing the same single instance, which you'd loop over for all students.

```
# select assignment group for quizzes to go into
examgroup  = c.getAssignmentGroup("Exams")

# write the quiz instructions.
```

(continues on next page)

(continued from previous page)

```
qdesc = (u'<h2>Stop!</h2>\n<p>By accessing this quiz, you are starting your␣
↪exam,'
        u' and the three hour window for submission.\xa0 Do\xa0<strong>not
↪</strong>'
        u' access the quiz before you are ready to begin.\xa0 If your␣
↪solution is'
        u' uploaded any time after the 3 hour window has expired, you will␣
↪receive'
        u' no credit for your exam.\xa0</p>\n<p>You do not need to submit␣
↪this quiz.'
        u' \xa0 All submission should be made to ' #insert link to␣
↪submission assignment here
        u' Your submission must be a\xa0<strong>single, clearly legible␣
↪PDF.'
        u' \xa0\xa0</strong>Do not submit individually scanned pages or␣
↪any other'
        u' format. You will only have one submission attempt, so be sure␣
↪to check your'
        u' work carefully before submitting.</p>')

# set due date and unlock date
duedate = c.localizeTime("2020-11-13")
unlockat = c.localizeTime("2020-11-11",duetime="09:00:00")

# generate quiz
quizdef = {
    "title": "Prelim Questions for %s"%nid,
    "description": qdesc,
    "quiz_type":"assignment",
    "assignment_group_id": examgroup.id,
    "time_limit": 180, #this is in minutes
    "shuffle_answers": False,
    "hide_results": 'always',
    "show_correct_answers": False,
    "show_correct_answers_last_attempt": False,
    "allowed_attempts": 1,
    "one_question_at_a_time": False,
    "published": False, #super important!
    "only_visible_to_overrides": True #super important!
    }
q1 = c.course.create_quiz(quiz=quizdef)

# add the payload to the quiz
purl = prelimupload[1]["url"]
pfname = prelimupload[1]["filename"]
pepoint = purl.split("/download")[0]
questext = (
    """<p>Your exam can be accessed here:"""
    """<a class="instructure_file_link instructure_scribd_file" title="{0}"
↪"""
    """href="{1}&amp;wrap=1" data-api-endpoint="{2}" """
    """data-api-returntype="File">{0}</a></p>\n<p>Don\'t worry if your␣
```

(continues on next page)

(continued from previous page)

```
↪browser """
    """warns you about navigating away from this page when trying to␣
↪download the """
    """exam - it won\'t break anything.</p>""".format(pfname, purl, pepoint)
)
quesdef = {
    "question_name": "Prelim",
    "question_text": questext,
    "question_type": "text_only_question",
}
q1.create_question(question=quesdef)

# now we can publish
q1.edit(quiz={"published":True})

# create assignment override to set due and unlock dates and the target␣
↪student
quizass = c.course.get_assignment(q1.assignment_id)
overridedef = {
    "student_ids":list(c.ids[c.netids == nid]),
    "title":'1 student',
    "due_at": duedate.strftime("%Y-%m-%dT%H:%M:%SZ"), #must be UTC
    "unlock_at": unlockat.strftime("%Y-%m-%dT%H:%M:%SZ"),
}
quizass.create_override(assignment_override=overridedef)
```

At the end of this process, you will have a quiz that is only assigned to the one student, and which will record the time at which the student accesses the exam PDF.

4. Once the exam period is over, you can now grab the submission times from the upload assignment as usual. To get the start times, you need to access the quiz submissions. Based on how we set this up, there should only be one submission per quiz, which makes things easier.

```
subtime = q1.get_submissions()[0].started_at_date

# if you want to re-localize the time:
from pytz import timezone
subtime.astimezone(timezone('US/Eastern'))
```

5. Here's some more stuff you can do in terms of post-processing. In this case, I have set up all of the quizzes and the students have completed their exams. I saved all of the student net ids (in a column labeled `netid`), assigned questions (for grading purposes), along with the quiz ids (in a column labeled `quizid`) in a CSV file called `assigned_questions.csv`. Now I can use that in order to access all of the individual start times, get all the end times and update the CSV file with this new info.

```
import pandas

dat = pandas.read_csv(os.path.join(prelimpath,'assigned_questions.csv'))

# loop through the quiz ids and get the start times
starts = []
for qid in dat['quizid'].values:
    print(qid)
```

(continues on next page)

(continued from previous page)

```
    q = c.course.get_quiz(qid)
    subs = q.get_submissions()
    try:
        starts.append(subs[0].started_at_date)
    except IndexError:
        starts.append(None)
starts = np.array(starts)

# now get the exam submission times
# change this to the name of your particular Exam assignment:
prelim = c.getAssignment('Prelim')
tmp = prelim.get_submissions()
subnetids = []
subtimes = []
for t in tmp:
    if t.user_id in c.ids:
        subnetids.append(c.netids[c.ids == t.user_id][0])
        if t.submitted_at:
            subtimes.append(datetime.strptime(
                t.submitted_at, """%Y-%m-%dT%H:%M:%S%z"""))
        else:
            subtimes.append(np.nan)
subnetids = np.array(subnetids)
subtimes = np.array(subtimes)

# now calculate each student's test duration
testtimes = []
subtimes2 = []
for j in range(len(dat)):
    try:
        testtimes.append((subtimes[subnetids == dat['netid'].values[j]][0] -
↪ starts[j]).seconds/3600)
        subtimes2.append(subtimes[subnetids == dat['netid'].values[j]][0])
    except TypeError:
        testtimes.append(np.nan)
        subtimes2.append(np.nan)
testtimes = np.array(testtimes)
subtimes = np.array(subtimes2)

# add info to CSV and write back to disk
dat = dat.assign(Start_Time=starts)
dat = dat.assign(End_Time=subtimes)
dat = dat.assign(Duration=testtimes)
dat.to_csv(os.path.join(prelimpath,'assigned_questions.csv'),index=False)
```

5.2 Split Assignments

A colleague wanted to give two different assignments to two sub-sections of their class (with a large enrollment, so doing it manually would be very annoying). They wanted everyone with an even netid to get one assignment, and everyone with an odd netid to get the other. Since the assignments could be deployed via quizzes, this can be done as a trivial extension of the example above: you generate two quizzes, and use assignment overrides to assign each one to half the course. It looks something like this:

```
from cornellGrading import cornellGrading
from datetime import datetime, timedelta
import numpy as np
import re

#set up  course
c = cornellGrading()
coursenum = ...
c.getCourse(coursenum)

#create two groups split by netid
netids = c.netids
names = c.names
canvasids = c.ids
netids = netids[names != 'Student, Test']
canvasids = canvasids[names != 'Student, Test']
names = names[names != 'Student, Test']
pn = re.compile('[a-z]+(\d+)')
numids = np.array([int(pn.match(n).group(1)) for n in netids])
odds = np.mod(numids,2).astype(bool)
groups = [list(canvasids[odds]), list(canvasids[~odds])]

#grab the assignment group you want this to go into
assgroup  = c.getAssignmentGroup("Assignments")

# set due date and unlock date
duedate = c.localizeTime("2021-09-08",duetime="10:00:00")
unlockat = c.localizeTime("2021-09-04",duetime="17:00:00")

# generate two quizzes
quiznames = ["Assignment 1a", "Assignment 1b"]
for j in range(1,3):
    quizdef = {
    "title": quiznames[j-1],
    "description": "Some text here",
    "quiz_type":"assignment",
    "assignment_group_id": assgroup.id,
    #"time_limit": 180, #this is in minutes
    "shuffle_answers": False,
    "hide_results": 'always',
    "show_correct_answers": False,
    "show_correct_answers_last_attempt": False,
    "allowed_attempts": 1,
    "one_question_at_a_time": False,
    "published": False, #super important!
```

(continues on next page)

(continued from previous page)

```
    "only_visible_to_overrides": True #super important!
    }
    q = c.course.create_quiz(quiz=quizdef)

    #can also add quiz payload here in form of pdf or whatever here

    # now we can publish
    q.edit(quiz={"published":True})

    # create assignment override to set due and unlock dates and the target␣
↪students
    quizass = c.course.get_assignment(q.assignment_id)
    overridedef = {
        "student_ids":groups[j-1],
        "title":'Group {}'.format(j),
        "due_at": duedate.strftime("%Y-%m-%dT%H:%M:%SZ"), #must be UTC
        "unlock_at": unlockat.strftime("%Y-%m-%dT%H:%M:%SZ"),
    }
    quizass.create_override(assignment_override=overridedef)
```

5.3 Managing Multiple Sections

As an alternative to the override-based strategy described above, assignments can be deployed in different ways (i.e., different due dates, or different assignment contents) to various subsections of a class by maintaining multiple course sections. Once again, for a large course, this is quite annoying to maintain via the web interface, but setting up a section with a specific set of students is very straightforward:

```
from cornellGrading import cornellGrading

c = cornellGrading()
coursenum = ...  # change to your actual number
c.getCourse(coursenum)

# create a new section called "Test Section 1"
sec = c.course.create_course_section(course_section={"name": "Test Section 1"})

# create array of userids to add to section.
# for example, to add everyone in the course:
uids = c.ids

for u in uids:
    sec.enroll_user(u)
```

Sections can also be deleted by executing `sec.delete()` on any section object `sec`. To remove a student from a section:

```
uid = ... #user id of student to remove

# get all enrollments in section
enrollments = sec.get_enrollments()
```

(continues on next page)

(continued from previous page)

```
# iterate through enrollments to find student to delete
for en in enrollments:
    if en.user['id'] == uid:
        en.deactivate("delete")
```

To get the object for a specific section:

```
secname = "Test Section 1"
secs = c.course.get_sections()

for sec in secs:
    if sec.name ==  secname:
        break
assert secname == sec.name
```

5.4 Uploading Qualtrics Results to Google Drive

When I use the self-grading approach enabled by *setupPrivateHW()* and *selfGradingImport()*, I like to assign one of my TAs or graders to do spot checks of students' self-assessments (note that it is equally important to look for students who are consistently undervaluing their work as those who are overvaluing their work). The grader obviously needs to see both the assignment submissions (available via Canvas SpeedGrader) as well as the individual question self-assessments. For the latter, I'd originally implemented the `sharewith` keyword for *setupPrivateHW()*, but using the Qualtrics web interface to do these spot checks proved to be overly tedious. However, in the process of calculating scores and uploading them to Canvas, *selfGradingImport()* pulls down a full spreadsheet of all user submissions. If we can automatically upload this to a shared Google Drive folder, that will make everyone's life much easier. Ok. Let's play the how many APIs can we tie together game?

The basics of Google Python API usage are given here: https://developers.google.com/drive/api/v3/quickstart/python. You need the `google-api-python-client`, `google-auth-httplib2`, and `google-auth-oauthlib` packages. Then you create a project and enable the relevant API, as described here: https://developers.google.com/workspace/guides/create-project. In particular, we're going to be using the Google Drive API Scopes: `.../auth/drive.file` and `.../auth/drive.metadata`. These will require you to create credentials and configure your OAuth consent screen, as described here: https://developers.google.com/workspace/guides/create-credentials. Create desktop application credentials and download the resulting JSON file. Note that while the documentation is ambiguous on this, you will need to add your own google account as a test user.

In addition to the credentials file, you'll need to create a token, which can similarly be saved to disk so that you only have to do the Google OAuth procedure once. Here's sample code, mostly based on the quickstart example (https://developers.google.com/drive/api/v3/quickstart/python):

```
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from googleapiclient.http import MediaFileUpload

SCOPES = ['https://www.googleapis.com/auth/drive.file',
          'https://www.googleapis.com/auth/drive.metadata']
```

(continues on next page)

(continued from previous page)

```
credfile = os.path.join(os.environ['HOME'], 'Downloads', 'credentials.json')
tokenfile = os.path.join(os.environ['HOME'], 'Downloads', 'token.json')

creds = None
if os.path.exists(tokenfile):
    creds = Credentials.from_authorized_user_file(tokenfile, SCOPES)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
    if creds and creds.expired and creds.refresh_token:
        creds.refresh(Request())
    else:
        flow = InstalledAppFlow.from_client_secrets_file(
            credfile, SCOPES)
        creds = flow.run_local_server(port=0)
    # Save the credentials for the next run
    with open(tokenfile, 'w') as token:
        token.write(creds.to_json())

service = build('drive', 'v3', credentials=creds)
```

Note that the token file is specific to the scopes in use. If you change scopes, you have to recreate the token from scratch. With this setup complete, all we need to do is find the folder we're want to put things in, and then grab and upload our spreadsheet. The following assumes that the folder is uniquely named in your Drive:

```
from cornellGrading import cornellGrading
c = cornellGrading()
coursenum = ...
c.getCourse(coursenum)
c.setupQualtrics()
hwnum = ...
res = c.selfGradingImport(hwnum, checkLate=True)

#find the Google Drive folder
tmp = service.files().list(q="name = 'Folder Name Goes Here'", spaces='drive',
                           fields='nextPageToken, files(id, name)',
                           pageToken=None).execute()
folderid = tmp['files'][0]['id']

#upload the file
media = MediaFileUpload(res[-1], resumable=True)
file = service.files().create(body={'name':'HW{} Self Assessments.csv'.
↪format(hwnum),
                                    'parents':[folderid]},
                              media_body=media, fields='id').execute()
```

And Bob's your uncle.

5.5 Repeated Assignments

Let's say you have an assignment that you wish to deploy to students multiple times. For example, you might wish to have weekly self-reflections or progress reports that can be collected via a Canvas quiz. Here's an easy way of setting this up.

Note: `canvasapi` does not yet have endpoint coverage for the 'Duplicate Assignment' action, so this code partially patches that functionality. The full endpoint coverage is expected to become part of the upstream code in a future release, at which point this code will be greatly simplified.

First, create the assignment you wish to clone, using either the API or the web interface. If using a Quiz (or New Quiz), build the whole quiz, including all questions. Assign the base assignment to whichever students or section you wish. For the sake of this example, we will call our base assignment 'Base Assignment'.

```
from cornellGrading import cornellGrading
from canvasapi.assignment import Assignment
from datetime import timedelta

c = cornellGrading()
coursenum = ...
c.getCourse(coursenum)

# grab the base assignment and any overrides associated with it:
ass = c.getAssignment("Base Assignment")
o1 = ass.get_overrides()[0]

# set the starting point for future due dates:
baseduedate = c.localizeTime("2023-10-06")

# now make as many copies as you wish.
# This is based on a 14 week semester:
for j in range(14):

    # duplicate the assignment
    response = ass._requester.request(
        "POST",
        "courses/{}/assignments/{}/duplicate".format(ass.course_id, ass.id),
    )
    newass = Assignment(ass._requester, response.json())
    newass.edit(assignment={"name": f"Weekly Update {j+1}"})

    # compute the new due date (+ j weeks at local noon)
    newduedate = c.localizeTime(
        (baseduedate + timedelta(days=7 * j)).strftime("%Y-%m-%d"),
        duetime="12:00:00",
    )

    # finally create an override for the assignment copy
    # this example unlocks the assignment one week prior to the due date
    overridedef = {
        "course_section_id": o1.course_section_id,
        "title": "espresso",
```

(continues on next page)

(continued from previous page)

```
        "due_at": newduedate.strftime("%Y-%m-%dT%H:%M:%SZ"),  # must be UTC
        "unlock_at": (newduedate + timedelta(days=-7)).strftime("%Y-%m-%dT%H:
↪%M:%SZ"),
    }
    newass.create_override(assignment_override=overridedef)
```

You can also get fancier and use a pre-determined set of due dates instead of a simple weekly scheme - all that you would need to do would be to modify the final for loop to read due dates from a file (or hard-coded array).

CHAPTER

SIX

CORNELLGRADING PACKAGE

6.1 Submodules

6.2 cornellGrading.cornellGrading module

class cornellGrading.cornellGrading.**cornellGrading**(*canvasurl='https://canvas.cornell.edu'*, *canvas_token_file=None*)

Bases: `object`

Class for io methods for Canvas and Qualtrics

Parameters
canvasurl (*str*) – Base url for canvas API calls. Defaults to https://canvas.cornell.edu

add2module(*module*, *title*, *obj*, *indent=0*, *position='bottom'*)

Adds an object to a module

Parameters

- **module** (*canvasapi.Module*) – The module to add the item to
- **title** (*str*) – Title of the module item
- **obj** (*canvasapi.CanvasObject*) – The object to be added to the module. Tested with Page and Assignment so far. Type should be one of [File, Page, Discussion, Assignment, Quiz, SubHeader, ExternalUrl, ExternalTool].
- **indent** (*int*) – Item indent level (defaults to zero)
- **position** (*str, int, or None*) – If None, add new item in the module's default position (typically at the bottom, but, confusingly, sometimes at the top). If a string, must be either 'top' or 'bottom' (case-isensitive). If int, this is interpreted as the desired position. Defaults to 'bottom'.

createAssignment(*name*, *groupid*, *submission_types=['none']*, *points_possible=10*, *published=True*, *description=None*, *allowed_extensions=None*, *due_at=None*, *unlock_at=None*, *external_tool_tag_attributes=None*)

Create an assignment

Parameters

- **name** (*str*) – Name of assignment.
- **groupid** (*int*) – Assignment group to put assignment in. Get group via getAssignmentGroup, and then use group.id attribute.

- **submission_types** (*list*) – See canvas API. Defaults to None.
- **points_possible** (*int*) – duh. If 0, will set grading_type to 'not_graded'.
- **published** (*bool*) – duh (defaults True)
- **description** (*str*) – The html assignment text. Not added if None (default).
- **allowed_extensions** (*list*) – List of strings for allowed extensions
- **due_at** (*datetime.datetime*) – Due date (not included if None). Must be timezone aware and UTC!
- **unlock_at** (*datetime.datetime*) – Unlock date (not included if None). Must be timezone aware and UTC!
- **external_tool_tag_attributes** (*dict*) – See API docs, which are incredibly unhelpful. Best to inspect an existing external tool assignment object

Returns
The assignment object

Return type
canvasapi.assignment.Assignment

Notes

https://canvas.instructure.com/doc/api/assignments.html#method.assignments_api.create

createAssignmentGroup(*groupName*)

Create assignment group by name

Parameters
assignmentGroup (*str*) – Name of assignment group to create. Cannot be name of existing group

Returns
The assignment group object

Return type
canvasapi.assignment.AssignmentGroup

createFolder(*folderName*, *parentFolder='course files'*, *hidden=False*)

Create folder by name

Parameters

- **folderName** (*str*) – Name of folder to create. Cannot be name of existing folder. Name can be nested (i.e., "Homework/HW1"), in which case createFolder will be called recursively until the folder is created. Do not put a leading slash on paths (they will always be relative to the parentFolder).
- **parentFolder** (*str*) – Name of parent folder to create in.
- **hidden** (*bool*) – Whether to toggle to hidden (false by default). If parent folder is hidden, you cannot make the subfolder visible.

Returns
The folder object

Return type
canvasapi.folder.Folder

Notes

Currently, all individual folder names should be unique (i.e., you shouldn't have "HW1" in multiple heirarchies, as all folder listing is flattened).

TODO: Fix this as soon as canvaspai wraps resolve_path https://github.com/ucfopen/canvasapi/issues/375

createPage(*title*, *body*, *editing_roles='teachers'*, *published=False*)

Create a Page

Parameters

- **title** (*str*) – Page title
- **body** (*str*) – Content of page (html formatted string)
- **editing_roles** (*str*) – See canvas API. Comma sepeated string, defaults to "teachers"
- **published** (*bool*) – Whether page is published on create (defaults True)

Returns
The new page object

Return type
canvasapi.page.Page

Notes

https://canvas.instructure.com/doc/api/pages.html#method.wiki_pages_api.create

If the title is the same as an existing page, Canvas will automatically append "-?" to the title, where ? is an incrementing integer.

dir2page(*path*, *title*, *extensions=None*, *prefix=''*, *folder='Lecture Notes'*, *hidden=True*, *editing_roles='teachers'*, *published=False*)

Generate new canvas page from a local dir with choice of file extensions

Parameters

- **path** (*str*) – Full path of the directory to process
- **title** (*str*) – Page title
- **extensions** (*list*) – List of strings for extensions to upload
- **prefix** (*str*) – Any HTML text to put before the links
- **folder** (*str*) – Canvas folder to upload the files or other supporting material to. Defaults to Images. If the folder does not exist, it will be created. See *cornellGrading.cornellGrading.createFolder()* for details.
- **hidden** (*bool*) – If the folder for image upload doesn't exist and needs to be created, it will have student visibility set by hidden. Defaults True (not visible to students without link).
- **editing_roles** (*str*) – See canvas API. Comma sepeated string, defaults to "teachers"
- **published** (*bool*) – Whether page is published on create (defaults False)

Returns
The new page object

Return type
canvasapi.page.Page

> **Warning:** Uploaded files will overwrite files of the same name in the upload folder.

genCourseMailingList()

Generates a qualtrics mailing list with all the netids from the course

Parameters
None –

Returns
None

genHWSurvey(*surveyname*, *nprobs*)

Create a HW self-grade survey

Parameters

- **surveyname** (*str*) – Name of survey
- **nprobs** (*int*) – Number of problems on the HW

Returns
Link url to the survey.

Return type
str

Notes

The survey will be created with a mandatory text entry field for the netid and then nprobs multiple choice fields for the problems with responses 0-4. The survey will be published and activated, so the link should be functional as soon as the method returns.

This survey will be public with an anonymous link.

> **Warning:** Deprecated since version 1.0.0: Use *cornellGrading.cornellGrading.genPrivateHWSurvey()* instead.

genPrivateHWSurvey(*surveyname*, *nprobs*, *scoreOptions=None*, *ecprobs=[]*)

Create a HW self-grade survey and make private

Parameters

- **surveyname** (*str*) – Name of survey
- **nprobs** (*int*) – Number of problems on the HW. Set to 0 for total score only.
- **scoreOptions** (*list of ints*) – Possible responses to each question. Defaults to 0,1,2,3
- **ecprobs** (*list of ints*) – Problems to be marked as extra credit (problem numbering starts at 1)

Returns
Unique survey ID

Return type
str

Notes

The survey will be created with nprobs multiple choice fields for the problems with responses 0-4. The survey will be published and activated, but made private. No distributions will be created.

getAssignment(*assignmentName*)

Locate assignment by name

Parameters
assignmentName (*str*) – Name of assignment to return. Must be exact match. To see all assignments do: >> for a in c.course.get_assignments(): print(a)

Returns
The Assignment object

Return type
canvasapi.assignment.Assignment

getAssignmentGroup(*groupName*)

Locate assignment group by name

Parameters
groupName (*str*) – Name of assignment group to return. Must be exact match. To see all assignments do: >> for a in c.course.get_assignment_groups(): print(a)

Returns
The assignment group object

Return type
canvasapi.assignment.AssignmentGroup

getCourse(*coursenum*)

Access course and load all student names, ids and netids

Parameters
coursenum (*int*) – Canvas course number to access. This can be looked up in Canvas, or you can look up all your courses:

```
>>> c = cornellGrading.cornellGrading()
>>> for cn in c.canvas.get_courses(): print(cn)
```

Returns
None

getFolder(*folderName*)

Locate folder by name

Parameters
(str) – Name of folder to return. Must be exact match. To see all assignments do: >> for a in c.course.get_folders(): print(a.name)

Returns
The folder object

Return type
canvasapi.folder.Folder

getGroups(*outfile=None*)

Create a csv file of group membership

Parameters
outfile (*str*) – Full path to output file. If not set, defaults to coursename Groups.csv in current directory.

Returns
None

Notes

This functionality is targeted at generating zoom breakout rooms, which is why the csv headers are what they are.

getModule(*moduleName*)

Locate module by name

Parameters
moduleName (*str*) – Name of module to return. Must be exact match. To see all assignments do: >> for a in c.listModules(): print(a)

Returns
The Module object

Return type
canvasapi.module.Module

getPage(*title*)

Locate page by title

Parameters
title (*str*) – Title of page to return. Must be exact match.

Returns
The page object

Return type
canvasapi.module.Module

latex2html(*fname*, *folder='Images'*, *hidden=True*)

Convert LaTex source into Canvas-compatible html and upload any required figures along the way

Parameters

- **fname** (*str*) – Full path of filename to process. If it has a PDF extension, assume that we're looking for the same filename .tex in the same directory. Otherwise, assumes that you're giving it the source file.
- **folder** (*str*) – Canvas folder to upload any images or other supporting material to. Defaults to Images. If the folder does not exist, it will be created. See createFolder for details.
- **hidden** (*bool*) – If the folder for image upload doesn't exist and needs to be created, it will have student visibility set by hidden. Defaults True (not visible to students without link).

Returns
List of strings of fully formatted html corresponding to the <body> block of a webpage

Return type
list

Notes

Requires pandoc to be installed and callable!

> **Warning:** Uploaded files will overwrite files of the same name in the upload folder.

latex2page(*fname*, *title*, *folder='Images'*, *hidden=True*, *editing_roles='teachers'*, *published=False*, *insertPDF=False*)

Generate a new canvas page out of a LaTex source file

Parameters

- **fname** (*str*) – Full path of filename to process. If it has a PDF extension, assume that we're looking for the same filename .tex in the same directory. Otherwise, assumes that you're giving it the source file.
- **title** (*str*) – Page title
- **folder** (*str*) – Canvas folder to upload any images or other supporting material to. Defaults to Images. If the folder does not exist, it will be created. See *cornellGrading.cornellGrading.createFolder()* for details.
- **hidden** (*bool*) – If the folder for image upload doesn't exist and needs to be created, it will have student visibility set by hidden. Defaults True (not visible to students without link).
- **editing_roles** (*str*) – See canvas API. Comma sepeated string, defaults to "teachers"
- **published** (*bool*) – Whether page is published on create (defaults False)
- **insertPDF** (*bool*) – If true, also include the original file in the page (this assumes that fname points at the compiled PDF and not the source).

Returns
The new page object

Return type
canvasapi.page.Page

Notes

Requires pandoc to be installed and callable!

> **Warning:** Uploaded files will overwrite files of the same name in the upload folder.

listAssignments()

Returns a list of assignments

Returns

asgnNames (list):
Matched ordered list of assignment strings (str list)

asgnIDs (list):
Matched ordered list of assignment IDs (int list)

Return type
tuple

`listCourses()`

Returns a list of courses

Returns

courseStrs (list):
Matched ordered list of course strings (str list)

courseNums (list):
Matched ordered list of course numbers (int list)

Return type
tuple

`listModules()`

List all modules in course

Parameters
`None` –

Returns
list of strings containing module names

Return type
list

`listPages()`

List all pages in course

Parameters
`None` –

Returns
list of strings containing page titles

Return type
list

`localizeTime`(*duedate*, *duetime='17:00:00'*, *tz='US/Eastern'*)

Helper method for setting the proper UTC time while being DST-aware

Parameters

- **duedate** (*str*) – Date in YYYY-MM-DD format
- **duetime** (*str*) – Time in HH-mm-SS format (default 17:00:00)
- **tz** (*str*) – pyzt timezone string (defaults to US/Eastern)

Returns
A time object. tzinfo will be <UTC>!

Return type
datetime.datetime

`matlabImport`(*assignmentNum*, *gradercsv*, *duedate*)

MATLAB grader grade import. Create the assignment in the MATLAB group and upload grades to it.

Parameters

- **assignmentNum** (*int*) – Number of assignment. Name will be "MATLAB N"
- **gradercsv** (*str*) – Full path to grader csv output.
- **duedate** (*str*) – Due date in format: YYYY-MM-DD (5pm assumed local time)

Returns
None

Notes

In assignment main page: Actions>Report. Choose 'Best solution as of today', Output:csv

If anyone was allowed a late submission, change the Late column entry to 'N'

> **Warning:** We are assuming that the timezone on your machine matches the timezone of the grader submitted time column. If there is a mismatch, there will be errors.

selfGradingImport(*assignmentNum*, *ecscore=3*, *checkLate=True*, *latePenalty=0.25*, *maxDaysLate=3*, *noUpload=False*, *saveDir=None*)

Qualtrics self-grading survey import.

Parameters

- **assignmentNum** (*int*) – Number of assignment. Name of survey will be "self.coursename HW# Self-Grade" Name of assignment will be HW#
- **escore** (*int*) – Extra credit score (defaults to 3)
- **checkLate** (*bool*) – Check for late submissions (defaults true)
- **latePenalty** (*float*) – Fraction of score to remove for lateness (defaults to 0.25). Must be in (0,1).
- **maxDaysLate** (*float*) – After this number of days past deadline, HW gets zero. Defaults to 3.
- **noUpload** (*bool*) – Don't upload if True (defaults False)
- **saveDir** (*str*) – Save path for raw survey output. Defaults to None (in which case it goes to the system tmp dir)

Returns

netids (str array):
Student netids

scores (float array):
Student scores

surveyfile (str):
Full path to survey download

Return type
tuple

Notes

To whitelist late submissions, in Canvas gradebook, click on the submission, click the right arrow, and then set status to 'None'.

setupHW(*assignmentNum*, *duedate*, *nprobs*)

Create qualtrics self-grading survey and Canvas column for a homework.

Parameters

- **assignmentNum** (*int*) – Number of assignment. Name of survey will be "self.coursename HW# Self-Grade" Name of assignment will be HW# Self-Grading
- **duedate** (*str*) – Due date in format: YYYY-MM-DD (5pm assumed local time)
- **nprobs** (*int*) – Number of howmework problems

Returns

None

Warning: Deprecated since version 1.0.0: Use *cornellGrading.cornellGrading.setupPrivateHW()* instead.

setupPrivateHW(*assignmentNum*, *nprobs*, *ecprobs=[]*, *sharewith=None*, *scoreOptions=None*, *createAss=False*, *solutions=None*, *injectText=False*, *selfGradeDueDelta=7*, *selfGradeReleasedDelta=3*)

Create qualtrics self-grading survey, individualized links distribution, a Canvas post for where the solutions will go, and injects links into assignment columns.

Parameters

- **assignmentNum** (*int*) – Number of assignment. Name of survey will be "self.coursename HW# Self-Grade" Name of assignment will be HW# Self-Grading
- **nprobs** (*int*) – Number of howmework problems
- **ecprobs** (*list of ints*) – Problems to be marked as extra credit (problem numbering starts at 1)
- **sharewith** (*str*) – Qualtrics id to share survey with. Defaults to None
- **scoreOptions** (*list of ints*) – Possible responses to each question. Defaults to 0,1,2,3
- **createAss** (*bool*) – Whether to create a self-grading assignment in Canvas (defaults False)
- **solutions** (*str*) – Full path to solutions file to upload.
- **injectText** (*bool*) – If True, will attempt to locate the tex file associated with the provided PDF in solutions (looking in the same directory), will then convert to Canvas compatible html using pandoc, and add to the assignment description. Requires pandoc to be installed and callable!
- **selfGradeDueDelta** (*float*) – Days after initial hw duedate that self-grading is due
- **selfGradeReleasedDelta** (*float*) – Days after initial hw duedate that self-grading (and solutions) are released.

Returns

None

setupQualtrics(*dataCenter='cornell.ca1'*, *qualtricsapi='.qualtrics.com/API/v3/'*, *qualtrics_token_file=None*)

Save/Load qualtrics api token and verify connection

Parameters

- **dataCenter** (*str*) – Root of datacenter url. Defaults to Cornell value.
- **qualtricsapi** (*str*) – API url. Defaults to v3 (current).
- **qualtrics_token_file** (*str*) – Full path to text file with qualtrics token on disk.

Returns
None

Notes

The dataCenter is the leading part of the qualtrics URL before "qualtrics.com" For an API token, on the qualtrics site, go to: Account Settings>Qualtrics IDs and click the 'Generate Token' button under API.

Warning: Windows users will have likely not be able to copy/paste this token into the command prompt. The best course is to use the qualtrics_token_file input, or just retype it into the prompt.

updateCourseMailingList()

Compares course qualtrics mailing list to current roster and updates as needed

Parameters
None –

Returns
None

uploadHW(*assignmentNum*, *duedate*, *hwfile*, *totscore=10*, *unlockDelta=None*, *injectText=False*, *allowed_extensions=None*, *moduleName=None*, *matlabParts=0*)

Create a Canvas assignment, set the duedae, upload an associated PDF and link in the assignment description, set the number of points, and (optionally), set an unlock time and inject the assignment text into the description along with the PDF link.

Parameters

- **assignmentNum** (*int*) – Number of assignment. Name of survey will be "self.coursename HW# Self-Grade" Name of assignment will be HW# Self-Grading
- **duedate** (*str*) – Due date in format: YYYY-MM-DD (5pm assumed local time)
- **hwfile** (*str*) – Full path to homework file
- **unlockDelta** (*float*) – Unlock this number of days before due date. Defaults to None, which makes the assignment initially unlocked.
- **injectText** (*bool*) – If True, will attempt to locate the tex file associated with the provided PDF in hwfile (looking in the same directory), will then convert to Canvas compatible html using pandoc, and add to the assignment description. Requires pandoc to be installed and callable!
- **allowed_extensions** (*list*) – List of strings for allowed extensions
- **moduleName** (*str*) – Name of module to add assignment object to.

- **matlabParts** (*int*) – If non-zero, this is a MATLAB assignment, and the relevant number of external tool Mathworks grader assignments will be created in the MATLAB Assignment group.

Returns
canvasapi.assignment.Assignment

Notes

All related files will be placed in path "Homeworks/HW?" where ? is the assignmentNum. Folders will be created as needed and will all be hidden.

uploadScores(*ass*, *netids*, *scores*)

Upload scores to Canvas

Parameters

- **ass** (*canvasapi.assignment.Assignment*) – Assignment object
- **netids** (*ndarray of str*) – Array of netids
- **scores** (*ndarray of floats*) – Array of scores matching ordering of netids

Returns
None

Notes

Only netids matching those found in the currently loaded course will be uplaoded.

waitForSubmit(*res*)

Wait for async result object to finish

6.3 cornellGrading.cornellInterface module

6.4 cornellGrading.cornellQualtrics module

class cornellGrading.cornellQualtrics.**cornellQualtrics**(*dataCenter='cornell.ca1'*, *qualtricsapi='.qualtrics.com/API/v3/'*, *qualtrics_token_file=None*)

Bases: object

Class for io methods for Qualtrics

Parameters

- **dataCenter** (*str*) – Root of datacenter url. Defaults to Cornell value.
- **qualtricsapi** (*str*) – API url. Defaults to v3 (current).

activateSurvey(*surveyId*)

Activate a Survey

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
None

addListContact(*mailingListId*, *firstName*, *lastName*, *email*)

Add a contact to a mailing list

Parameters

- **mailingListId** (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId
- **firstName** (*str*) – First name
- **lastName** (*str*) – duh
- **email** (*str*) – double duh

Returns
None

Notes:

addSurveyQuestion(*surveyId*, *questionDef*)

Add question to existing Survey

Parameters

- **surveyId** (*str*) – Survey ID string as returned by getSurveyId
- **questionDef** (*dict*) – Full question definition dictionary

Returns
Question ID

Return type
str

addSurveyQuota(*surveyId*, *quotaDef*)

Add quota to existing Survey

Parameters

- **surveyId** (*str*) – Survey ID string as returned by getSurveyId
- **quotaDef** (*dict*) – Full question definition dictionary

Returns
Quota ID

Return type
str

addSurveyQuotaGroup(*surveyId*, *quotaGroupName*)

Add quota to existing Survey

Parameters

- **surveyId** (*str*) – Survey ID string as returned by getSurveyId
- **quotaDef** (*dict*) – Full question definition dictionary

Returns
Quota Group ID

Return type
str

createLibraryMessage(*libraryId*, *subject*, *msg*, *category='thankYou'*)

Create message in a library

Parameters

- **libraryId** (*str*) – Library ID string
- **subject** (*str*) – Message description
- **msg** (*str*) – Message text
- **category** (*str*) – Message category: invite, inactiveSurvey, reminder, thankYou, endOfSurvey, general, validation, lookAndFeel, emailSubject, smsInvite

Returns
message id

Return type
str

createReminderDistribution(*distributionId*, *libraryId*, *messageId*, *subject*, *sendDate*, *replyTo='no-reply@cornell.edu'*, *fromName='A Survey Robot'*)

Create a survey reminder distribution

Parameters

- **distributionId** (*str*) – Unique id string of existing distribution
- **libraryId** (*str*) – Unique id string of library
- **messageId** (*str*) – Unique id string of reminder message
- **subject** (*str*) – Subject line
- **sendDate** (*datetime.datetime*) – Send date
- **replyTo** (*str*) – Reply-to address
- **fromName** (*str*) – From string

Returns
distribution id

Return type
str

Notes:

createSingleContactDistribution(*surveyId*, *mailingListId*, *contactId*, *subject*, *cmsg=''*, *replyTo='no-reply@cornell.edu'*)

Create a survey distribution for the given mailing list

Parameters

- **surveyId** (*str*) – Unique id string of survey. Get either from web interface or via getSurveyId
- **mailingListId** (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId
- **contactId** (*str*) – Unique id string of contact. Get either from we interface or via getMailingList
- **subject** (*str*) – Subject line
- **cmsg** (*str*) – Custom message to add to standard email

- **replyTo** (*str*) – Reply-to address

Returns
distribution id

Return type
str

Notes:

createSurvey(*surveyname*)

Create a new survey

Parameters
surveyname (*str*) – Name of survey

Returns
Unique survey id

Return type
str

Notes

Adapted from https://api.qualtrics.com/reference#create-survey English and ProjectCategory: CORE are hard-coded. Qualtrics will allow you to create multiple surveys with the same name, but we would like to enforce uniqueness so this is explicitly dissallowed by the method.

deleteListContact(*mailingListId*, *contactId*)

Add a contact to a mailing list

Parameters

- **mailingListId** (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId
- **contactId** (*str*) – Unique id string of contact to remove (as returned by getListContacts)

Returns
None

Notes:

deleteSurvey(*surveyId*)

Delete a Survey

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
None

dumpContacts(*mailingListId*)

Repackage all contacts in a mailing list into a pandas DataFrame

Parameters
mailingListId (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId

Returns
dataframe with all mailing list contacts

Return type
pandas.DataFrame

Notes:

exportSurvey(*surveyId*, *fileFormat='csv'*, *useLabels='true'*, *saveDir=None*)

Download and extract survey results

Parameters

- **surveyId** (*str*) – Unique id string of survey. Get either from web interface or via getSurveyId
- **fileFormat** (*str*) – Format to download (must be csv, tsv, or spss)
- **useLabels** (*str*) – Use choice labels ("true" or "false", defaults "true")
- **saveDir** (*str*) – Full path to save location. If None (default) uses system tmp dir

Returns
Full path to directory where unzipped file will be. Filename should be the same as the survey name except with any colons replaced with underscores

Return type
str

Notes

Adapted from https://api.qualtrics.com/docs/getting-survey-responses-via-the-new-export-apis

genDistribution(*surveyId*, *mailingListId*)

Create a survey distribution for the given mailing list

Parameters

- **surveyId** (*str*) – Unique id string of survey. Get either from web interface or via getSurveyId
- **mailingListId** (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId

Returns
Dicts containing unique links ['link'] for each person in the mailing list ['email']

Return type
list

Notes:

genMailingList(*listName*)

Generate mailing list

Parameters
listName (*str*) – List name

Returns
Unique list id

Return type
str

getDistributionLinks(*distributionId*, *surveyId*)

Retrieve distribution info

Parameters

- **distributionId** (*str*) – Unique id string of distribution.
- **surveyId** (*str*) – Unique id string of survey.

Returns
dicts of surveylinks for all members of original contact list used

Return type
list

Notes

Only the links of those people actually included in the distribution will be real

getLibraryMessage(*libraryId*, *messageId*)

get message from a library

Parameters

- **libraryId** (*str*) – Library ID string
- **messageId** (*str*) – Message ID string

Returns
message dictionaries

Return type
list

getListContacts(*mailingListId*)

Get all contacts in a mailing list

Parameters
mailingListId (*str*) – Unique id string of mailing lists. Get either from we interface or via getMailingListId

Returns
list of dicts encoding all mailing list contacts

Return type
list

Notes:

getMailingListId(*listName*)

Find qualtrics mailinglist id by name. Matching is exact.

Parameters
listName (*str*) – Exact text of list name

Returns
Unique list id

Return type
str

`getMailingLists()`

Grab all available Qualtrics mailing lists

Parameters
`None` –

Returns
response object with all mailing lists

Return type
requests.models.Response

`getSurvey(surveyId)`

Get a Survey

Parameters
`surveyId` (*str*) – Survey ID string as returned by getSurveyId

Returns
Dictionary of survey (response.json()['result'])

Return type
dict

`getSurveyId(surveyname, renew=False)`

Find qualtrics survey id by name. Matching is exact.

Parameters

- `surveyname` (*str*) – Exact text of survey name
- `renew` (*bool*) – Renew stored list of surveys (default False)

Returns
Unique survey id

Return type
str

`getSurveyNames(renew=False)`

Return a list of all current survey names.

Parameters
`renew` (*bool*) – Renew stored list of surveys (default False)

Returns
All survey names

Return type
list

`getSurveyOptions(surveyId)`

Return survey options

Parameters
`surveyId` (*str*) – Survey ID string as returned by getSurveyId

Returns
Survey options data

Return type
dict

`getSurveyQuestions`(*surveyId*)

Grab all available survey questions

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
The response object with the questions.

Return type
requests.models.Response

`getSurveyQuotaGroups`(*surveyId*)

Get a Survey's quota groups

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
Quota ids (response.json()['result']["elements"][0]["Quotas"])

Return type
list

`getSurveyQuotas`(*surveyId*)

Get all quotas for a survey

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
Dictionary of quotas (response.json()['result'])

Return type
dict

`listDistributions`(*surveyId*)

Get all survey distribution ids for the given survey ID

Parameters
surveyId (*str*) – Unique id string of survey. Get either from web interface or via getSurveyId

Returns
Dicts containing distributions

Return type
list

Notes:

`listLibraries`()

List all libraries

Parameters
None –

Returns
Dictionary of survey (response.json()['result'])

Return type
dict

listLibraryMessages(*libraryId*)

list messages in a library

Parameters
libraryId (*str*) – Library ID string

Returns
message dictionaries

Return type
list

listSurveys(*baseUrl=None*)

Grab and store all available Qualtrics surveys

Parameters
None –

Returns
surveynames (list) surveyids (list)

Return type
tuple

makeSurveyPrivate(*surveyId*)

Make a Survey private

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
None

publishSurvey(*surveyId*)

Publish a Survey

Parameters
surveyId (*str*) – Survey ID string as returned by getSurveyId

Returns
None

setupHeaders()

Generate standard headers

Parameters
None –

Returns
None

shareSurvey(*surveyId*, *sharewith*)

Share survey with another qualtrics user :param surveyId: Unique survey id string :type surveyId: str :param sharewith: Qualtrics id to share survey with :type sharewith: str

Returns
None

Notes:

updateSurveyOptions(*surveyId*, *data*)

Return survey options

Parameters

- **surveyId** (*str*) – Survey ID string as returned by getSurveyId
- **data** (*dict*) – Dictionary of options (as returned by getSurveyData)

Returns
None

updateSurveyQuestion(*surveyId*, *qId*, *questionDef*)

Add question to existing Survey

Parameters

- **surveyId** (*str*) – Survey ID string as returned by getSurveyId
- **qId** (*str*) – Question ID string as returned by addSurveyQuestion
- **questionDef** (*dict*) – Full question definition dictionary

Returns
None

6.5 cornellGrading.dueDatesFromCSV module

6.6 cornellGrading.pandocHTMLParser module

class cornellGrading.pandocHTMLParser.**pandocHTMLParser**(*hwd*, *upfolder*)

Bases: HTMLParser

Parser for pandoc produced html from LaTeX source

handle_data(*data*)

handle_endtag(*tag*)

handle_starttag(*tag*, *attrs*)

6.7 cornellGrading.upload_MC_questions module

6.8 Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

H

L

M

P

S

U

W