pl-cee202-docs

Release 0.0.0

Apr 29, 2022

Students

Author: Dr. Zhonghua Zheng (zhonghua.zheng@outlook.com)

This website provides the information about how to set up and use PrairieLearn tailored for CEE 202 - Engineering Risk & Uncertainty.

PrairieLearn is an online problem-driven learning system for creating homeworks and tests.

CHAPTER 1

Table of Contents

1.1 How to Access PrairieLearn?

- Access PrairieLearn by going to: https://prairielearn.engr.illinois.edu
- Enter your netID and password
- You're in!

1.2 How to Run PrairieLearn?

For more information, please check HERE.

1.2.1 Install and check (for first time users)

- **Step 1**: Install Docker Community Edition.
- **Step 2**: Open a terminal, such as iTerm2 (on **MacOS/Linux**), and run PrairieLearn using the example course with

```
docker run -it --rm -p 3000:3000 prairielearn/prairielearn
```

- **Step 3**: Open a web browser and connect to http://localhost:3000/pl
- **Step 4**: Click the button **`Load from disk`** in the upper right, then play with it.
- **Step 5**: When you are finished with PrairieLearn, type `Control-C` on the commandline where your ran the server to stop it.

1.2.2 Routine work

- **Step 1**: Upgrade your Docker's version on PrairieLearn

```
docker pull prairielearn/prairielearn
```

- **Step 2**: To use your own course, use the `-v` flag to bind the Docker `/course` directory with your own course directory (replace the precise path with your own).

  on **Windows**:

```
docker run -it --rm -p 3000:3000 -v C:\GitHub\pl-cee202:/course prairielearn/
↪prairielearn
```

  or on **MacOS/Linux**:

```
docker run -it --rm -p 3000:3000 -v /Users/zzheng25/git/pl-cee202:/course␣
↪prairielearn/prairielearn
```

- **Step 3**: Open a web browser and connect to http://localhost:3000/pl
- **Step 4**: Click the button **`Load from disk`** in the upper right, then work on it.
- **Step 5**: When you are finished with PrairieLearn, type `Control-C` on the commandline where your ran the server to stop it.

1.3 Numerical Answers with R

This page is specific to the **R** questions (**without coding**). The objectives are:

- Use the necessary R function in the `server.py` to generate the solutions, and grade the questions
- Specify the randomized variables in the `server.py`
- Specify the specific files (e.g., **figure**) in the `server.py`

1.3.1 Overview

The easiest way to create a R question (without coding) is by copying an existing R question, and change certain files. Then you don't need to create the UUID by yourself.

Note: Each UUID will be assigned to a question only.

1.3.2 Step 1: Copy a R question

- Follow the Step 1 to Step 4 in the **Routine work**. Then click **PrarieLearn** logo (next to **Admin**) in the upper left.
- Click a course such as **`CEE 202: Engineering Risk & Uncertainty`** in the `Courses` (not `Course instances`) list.
- Click the `Questions` (next to `Issues`) on the top line.
- Find a question you want to copy (for example: `AS4_Prob5_2020_AngTang`).
- Click `Settings` between `Preview` and `Statistics`.
- Click `Make a copy of this question`

- Click `Change QID`

1.3.3 Step 2: Modify the questions

Before you modifying the question, I strongly suggest creating a **spreadsheet** to keep track of the questions (including title, topic, tags) and corresponding UUID.

Note: Each question folder contain the following files

info.json

- Click `Edit` under `Settings`
- Define the `title`, `topic`, `tags`, and `type`

server.py

- Click `Files` (under `PrairieLearn` in the upper left) → `Edit` the `server.py`, then you need to finish the following tasks:

```
import rpy2.robjects as robjects
import prairielearn as pl

def generate(data):
    # here is the start the R function
    values = robjects.r("""
    # prob 1
    #a_r = 4.0
    a_r = sample(seq(3.8,4.3,0.1),1)
    ans_a_r = 1 + a_r

    # Export
    list(
         ans = list(a=a_r,
                    answer_a=ans_a_r)
       )
    """)
    # here is the end of the R function
    ans = values[0]
    # Convert from R lists to python dictionaries
    ans = { key : ans.rx2(key)[0] for key in ans.names }
    # Setup output
    data['correct_answers'] = ans
    # Setup randomized variables
    data["params"] = ans
    # define the figure name
    image_name = "dist.png"
    data["params"]["image"] = image_name
```

- Change the randomized variable using `a_r=sample(seq(start,end,interval),1)`
- Change the answers (`ans_a_r`, `ans_b_r`, ...), and export (`list(...)`)

Note: `a` corresponds to `${{params.a}}$`, `answer_a` corresponds to `answers-name="answer_a"` in the `question.html`

- Change the `image_name` (if you have figures(s))

question.html

- Click `Files` (under `PrairieLearn` in the upper left) → `Edit` the `question.html`, then you need to finish the following tasks:

```
<pl-question-panel>
        <p>
        This is the problem statement.
        </p>
        <pl-figure file-name={{params.image}} directory="clientFilesQuestion"></pl-
↪figure>
</pl-question-panel>

<pl-question-panel><hr></pl-question-panel>
<pl-question-panel>
        <p>
            (a) Determine the probability that the settlement will exceed ${{params.a}
↪}$ cm.
        </p>
</pl-question-panel>

<div class="card my-2">
  <div class="card-body">
    <pl-question-panel>
      <p>The answer is: (0.XX)</p>
    </pl-question-panel>
    <pl-number-input answers-name="answer_a" weight = "3" comparison="relabs" rtol="0.
↪01" atol="0.01"></pl-number-input>
  </div>
</div>
```

- Replace "This is the problem statement." with your **problem statement**
- Replace `${{params.a}}$` with your randomized variable from `server.py`
- Replace `"answer_a"` with your answer from `server.py`
- Define the tolerance. Sotiria suggests that:
  - for the answer (0.XX), `comparison="relabs" rtol="0.01" atol="0.01"`
  - for the answer (0.XXX), `comparison="relabs" rtol="0.001" atol="0.001"`

1.3.4 Alternatives: Integer

Reference: (link)

If the answer is an **integer**, you need to replace

```
<pl-number-input answers-name="answer_a" weight = "3" comparison="relabs" rtol="0.01"␣
↪atol="0.01"></pl-number-input>
```

with

```
<pl-integer-input answers-name="answer_a" weight = 3></pl-integer-input>
```

Where the answer "answer_a" has to be an **integer**.

1.3.5 Step 3: Test your questions

- Click `Preview` to test
- Click `New variant` to have another test

1.3.6 Step 4: Commit and push the changes

- Using Git to commit and push the changes

Note: You may do this after you finish all the questions

1.3.7 Step 5: Sync and test

- Log in the website https://prairielearn.engr.illinois.edu/pl/, and select your course
- Click `Sync`, then `Pull from remote git repository`
- Find your questions by clicking `Questions` and test them again

1.3.8 Appendix: Answers from R function output

The following `server.py` shows the workflow of doing a simple linear regression

```
import rpy2.robjects as robjects
import prairielearn as pl

def generate(data):
    # here is the start the R function
    values = robjects.r("""

    # Read in the data
    my_data = read.csv(paste0('./clientFilesQuestion/mydata.csv'))

    # Form a model
    lm_model = lm(y ~ x, data = my_data)
    beta_hats = coef(lm_model)

    # View all the attributes, e.g., adj.r.squared
    #attributes(summary(lm_model))

    # New predictions
    #new_data <- data.frame(x = c(20))
    #predicted = predict(lm_model,newdata=new_data,interval="confidence", level=0.95)

    # Export
    list(
         ans = list(beta1 = beta_hats[2],
                    beta0 = beta_hats[1],
                    pvalue = summary(lm_model)$coefficients[2,4],
                    slope_se = summary(lm_model)$coefficients[2,2],
                    r_mult=summary(lm_model)$r.squared,
                    lo=confint(lm_model)[2,1],
                    hi=confint(lm_model)[2,2],
                    pred_low = predicted[2],
```

(continues on next page)

(continued from previous page)

```
                pred_high = predicted[3],
                corr = cor(x,y))
    )
"""")
# Extract parameter and answer lists
ans = values[0]
# Convert from R lists to python dictionaries
ans = { key : ans.rx2(key)[0] for key in ans.names }
# Setup output dictionaries
data['correct_answers'] = ans
```

1.4 Numerical Answers with Python

This page is specific to the **Python** questions (**without coding**). The objectives are:

- Use the necessary Python function in the `server.py` to generate the solutions, and grade the questions
- Specify the randomized variables in the `server.py`
- Specify the specific files (e.g., **figure**) in the `server.py`

1.4.1 Overview

The easiest way to create a Python question (without coding) is by copying an existing question (either Python or R), and change certain files. Then you don't need to create the UUID by yourself.

Note: Each UUID will be assigned to a question only.

1.4.2 Step 1: Copy a question

- Follow the Step 1 to Step 4 in the **Routine work**. Then click **PrarieLearn** logo (next to **Admin**) in the upper left.
- Click a course such as **`CEE 202: Engineering Risk & Uncertainty`** in the `Courses` (not `Course instances`) list.
- Click the `Questions` (next to `Issues`) on the top line.
- Find a question you want to copy (for example: `AS4_Prob5_2020_AngTang`).
- Click `Settings` between `Preview` and `Statistics`.
- Click `Make a copy of this question`
- Click `Change QID`

1.4.3 Step 2: Modify the questions

Before you modifying the question, I strongly suggest creating a **spreadsheet** to keep track of the questions (including title, topic, tags) and corresponding UUID.

Note: Each question folder contain the following files

info.json

- Click `Edit` under `Settings`
- Define the `title`, `topic`, `tags`, and `type`

server.py

- Click `Files` (under `PrairieLearn` in the upper left) → `Edit` the `server.py`, then you need to finish the following tasks:

```
import rpy2.robjects as robjects
import prairielearn as pl
import numpy as np
from numpy import arange
from numpy.random import choice

def generate(data):
    # start to code your solution
    ## "a" could be 3.8, 3.9, 4.0, 4.1, 4.2
    a = choice(arange(3.8,4.3,0.1),1)
    answer_a = a+1
    # here is the end of your solution

    # Setup output
    data['correct_answers']["answer_a"] = answer_a
    # Setup randomized variables
    data["params"]["a"] = a
    # define the figure name
    image_name = "dist.png"
    data["params"]["image"] = image_name
```

- import the necessary packages at the beginning
- Change the randomized variable using `a=choice(arange(start,end+interval,interval),1)`

Note: `a` corresponds to `${{params.a}}$`, `answer_a` corresponds to `answers-name="answer_a"` in the `question.html`

- Change the `image_name` (if you have figure(s))

question.html

- Click `Files` (under `PrairieLearn` in the upper left) → `Edit` the `question.html`, then you need to finish the following tasks:

```
<pl-question-panel>
        <p>
        This is the problem statement.
        </p>
        <pl-figure file-name={{params.image}} directory="clientFilesQuestion"></pl-
↪figure>
</pl-question-panel>

<pl-question-panel><hr></pl-question-panel>
<pl-question-panel>
        <p>
```

(continues on next page)

(continued from previous page)

```
            (a) Determine the probability that the settlement will exceed ${{params.a}
↪}$ cm.
        </p>
</pl-question-panel>

<div class="card my-2">
  <div class="card-body">
    <pl-question-panel>
      <p>The answer is: (0.XX)</p>
    </pl-question-panel>
    <pl-number-input answers-name="answer_a" weight = "3" comparison="relabs" rtol="0.
↪01" atol="0.01"></pl-number-input>
  </div>
</div>
```

- Replace "This is the problem statement." with your **problem statement**
- Replace `${{params.a}}$` with your randomized variable from `server.py`
- Replace `"answer_a"` with your answer from `server.py`
- Define the tolerance. Sotiria suggests that:
  - for the answer (0.XX), `comparison="relabs" rtol="0.01" atol="0.01"`
  - for the answer (0.XXX), `comparison="relabs" rtol="0.001" atol="0.001"`

1.4.4 Alternatives: Integer

Reference: (link)

If the answer is an **integer**, you need to replace

```
<pl-number-input answers-name="answer_a" weight = "3" comparison="relabs" rtol="0.01"␣
↪atol="0.01"></pl-number-input>
```

with

```
<pl-integer-input answers-name="answer_a" weight = 3></pl-integer-input>
```

Where the answer "answer_a" has to be an **integer**.

1.4.5 Step 3: Test your questions

- Click `Preview` to test
- Click `New variant` to have another test

1.4.6 Step 4: Commit and push the changes

- Using Git to commit and push the changes

Note: You may do this after you finish all the questions

1.4.7 Step 5: Sync and test

- Log in the website https://prairielearn.engr.illinois.edu/pl/, and select your course
- Click `Sync`, then `Pull from remote git repository`
- Find your questions by clicking `Questions` and test them again

1.5 Multiple Choice

Reference: (link)

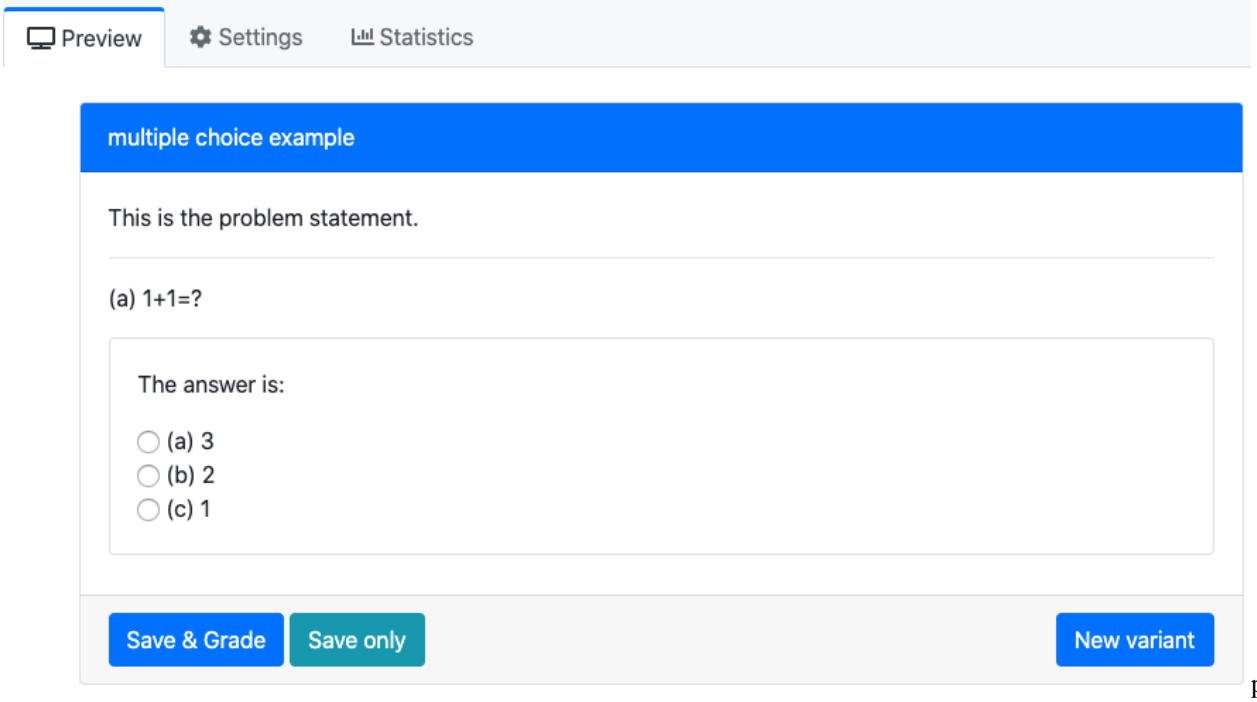
The multiple choice only requires you to modify:

A `pl-multiple-choice` element selects **one** correct answer and zero or more incorrect answers and displays them in a random order as radio buttons.

1.5.1 An example of the `question.html`

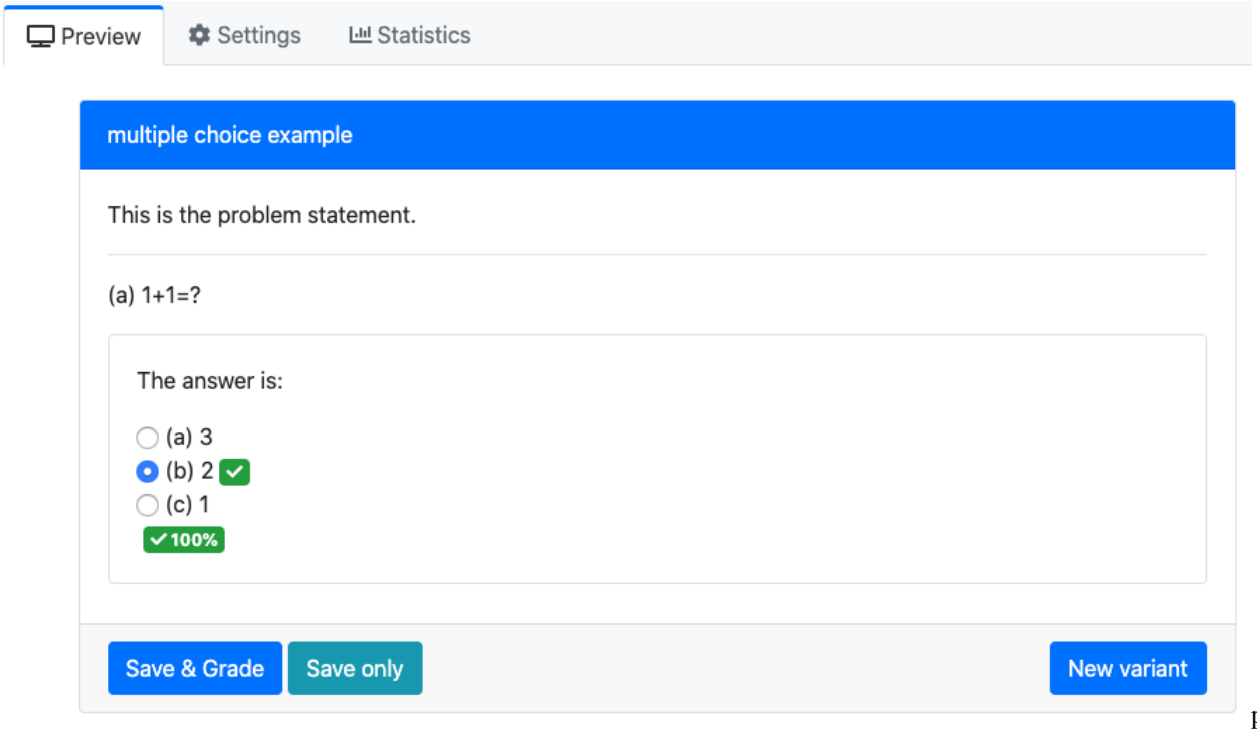
```
<pl-question-panel>
        <p>
        This is the problem statement.
        </p>
</pl-question-panel>

<pl-question-panel><hr></pl-question-panel>
<pl-question-panel>
        <p>
            (a) 1+1=?
        </p>
</pl-question-panel>

<div class="card my-2">
    <div class="card-body">
        <pl-question-panel>
            <p>
                The answer is:
            </p>
        </pl-question-panel>

        <pl-multiple-choice answers-name="answer_a" weight="2">
                <pl-answer correct="false"> 1 </pl-answer>
                            <pl-answer correct="false"> 3 </pl-answer>
                            <pl-answer correct="true"> 2 </pl-answer>
        </pl-multiple-choice>
    </div>
</div>
```

The problem will be:



pl-multiple-choice

When you clicked the correct answer:



pl-multiple-choice

1.5.2 Conditional Answers

Assume we want to have conditional answers, for instance, the answers of the multiple choice depend on the previous answer. Here we have an example, the `p-value` is calculated from previous answer (we omiss how to get `p`, but use the function `sample` as an example). Here the `p` (in Python is p, in R is p_r, use the function `ans=list(...)` to convert) value could be `0.5` or `0.005`. The idea is:

If p<0.01, the correct answer is True (reject), and vice versa.

server.py

Please note the order for the conditional answers, otherwise the commands `data['correct_answers'] = ans` and `data["params"] = ans` will overwrite your conditional answers.

```
import prairielearn as pl
def generate(data):
    values = robjects.r("""
        p_r = sample(c(0.005,0.5),1)

        # Export
        list(
             ans = list(p=round(p_r,digits=3))
       )
    """)

    ans = values[0]
    # Convert from R lists to python dictionaries
    ans = { key : ans.rx2(key)[0] for key in ans.names }
    # Setup output dictionaries
    data['correct_answers'] = ans
    data["params"] = ans

    # Here is the start for the conditional answers
    if data['correct_answers']["p"]<0.01:
        # The option "True" in question.html is correct
        data['params']["answer_b_true"] = True
        data['params']["answer_b_false"] = False
    else:
        # The option "True" in question.html is incorrect
        data['params']["answer_b_true"] = False
        data['params']["answer_b_false"] = True
```

question.html

```
<pl-question-panel><hr></pl-question-panel>
<pl-question-panel>
    <p>
        (b) If the p-value is ${{params.p}}$, we should reject $H_0$
    </p>
</pl-question-panel>

<div class="card my-2">
    <div class="card-body">
        <pl-question-panel>
            <p>
```

(continues on next page)

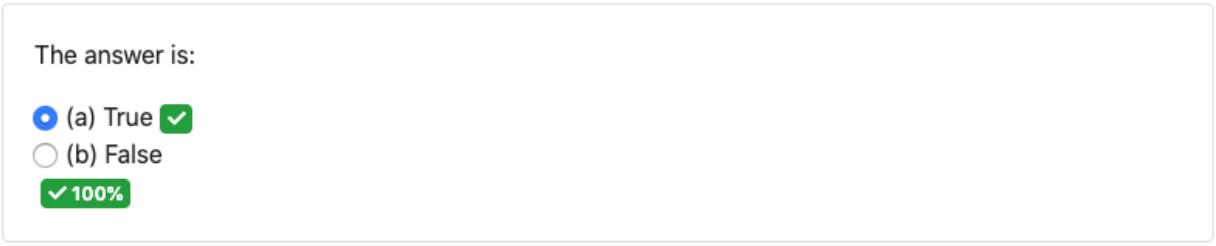
(continued from previous page)

```
                The answer is:
            </p>
        </pl-question-panel>

        <pl-multiple-choice answers-name="answer_b" weight="2">
          <pl-answer correct="{{params.answer_b_true}}"> True </pl-answer>
          <pl-answer correct="{{params.answer_b_false}}"> False </pl-answer>
        </pl-multiple-choice>
    </div>
</div>
```

Appearance

- If p-value is `0.005`

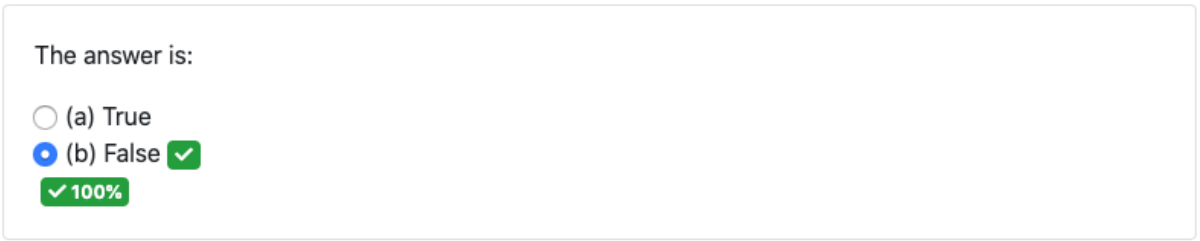
(b) If the p-value is 0.005, we should reject H_0



pl-multiple-choice

- If p-value is `0.5`

(b) If the p-value is 0.5, we should reject H_0



pl-multiple-choice

1.5.3 Customizations

Inside the `pl-multiple-choice` element, each choice must be specified with a `pl-answer` that has attributes:

1.6 Checkbox

Reference: (link)

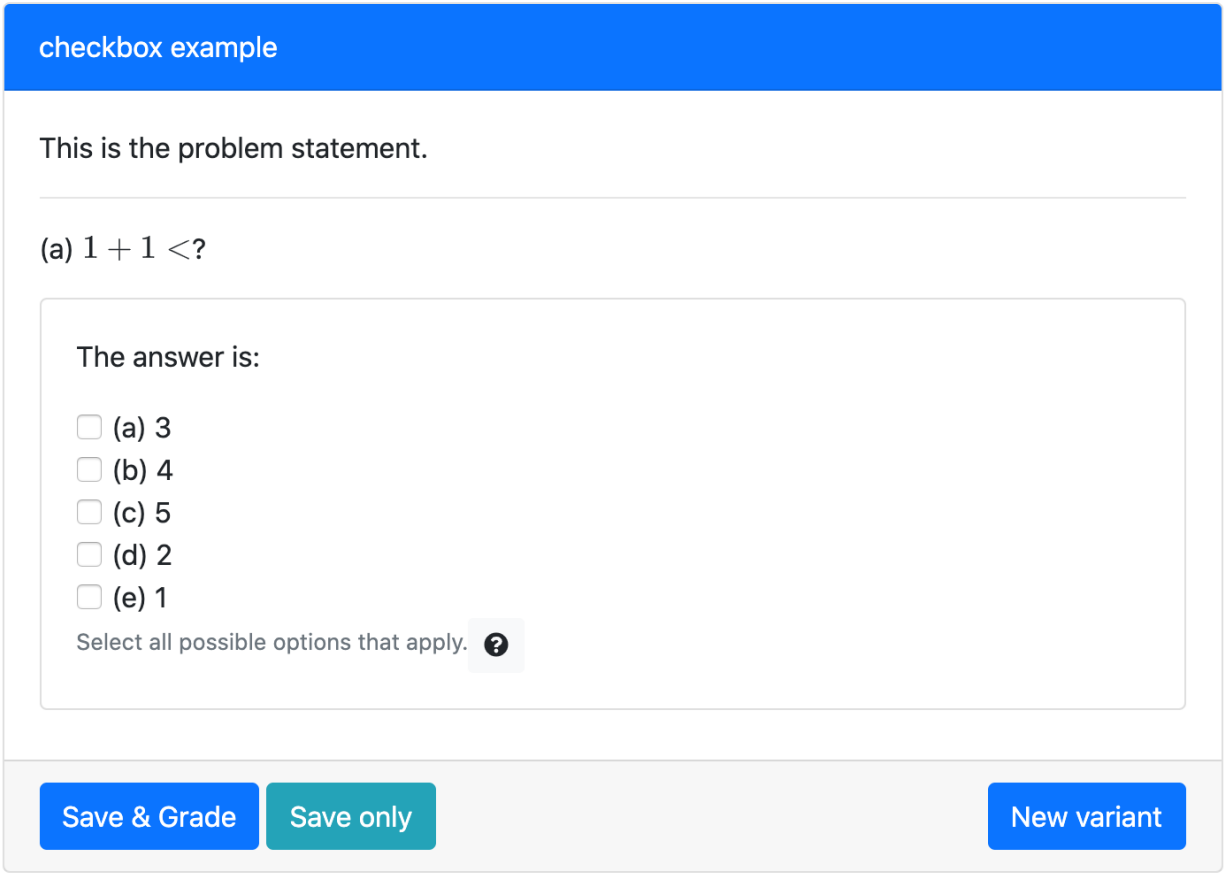
The checkbox only requires you to modify:

A `pl-checkbox` allows for **one** or **more** choices. It displays a subset of the answers in a random order as checkboxes.

1.6.1 An example of the `question.html`

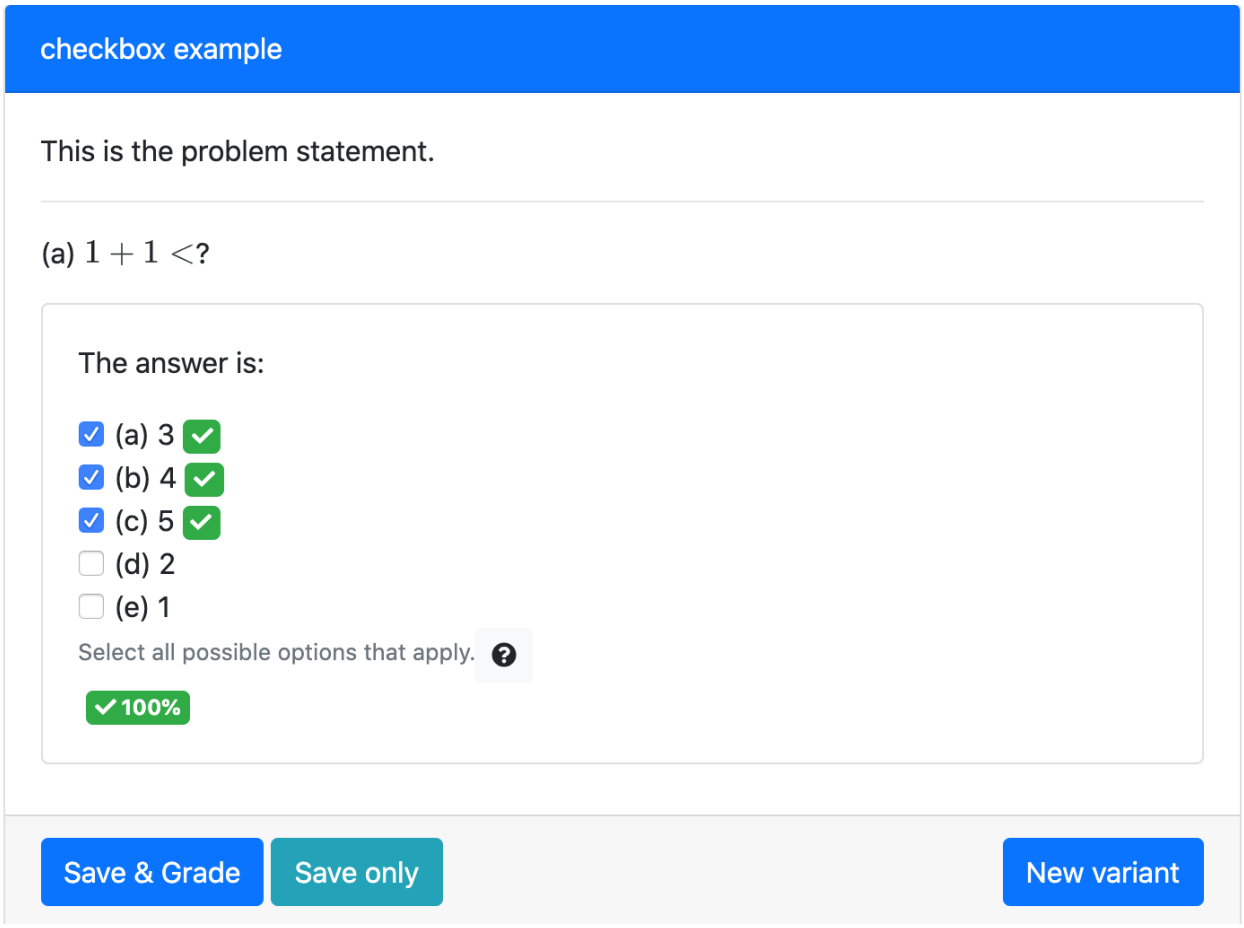
```
<pl-question-panel>
        <p>
        This is the problem statement.
        </p>
</pl-question-panel>

<pl-question-panel><hr></pl-question-panel>
<pl-question-panel>
        <p>
            (a) $1+1<$?
        </p>
</pl-question-panel>

<div class="card my-2">
    <div class="card-body">
        <pl-question-panel>
            <p>
                The answer is:
            </p>
        </pl-question-panel>

        <pl-checkbox answers-name="vpos" weight="1">
          <pl-answer correct="true">5</pl-answer>
          <pl-answer correct="true">4</pl-answer>
          <pl-answer>               1</pl-answer>
          <pl-answer correct="true">3</pl-answer>
          <pl-answer>               2</pl-answer>
        </pl-checkbox>
    </div>
</div>
```

The problem will be:



pl-multiple-choice

When you clicked the correct answer:



pl-multiple-choice

1.7 Another example question R autograder

By Neetesh Sharma (Department of CEE, University of Illinois, Urbana-Champaign, IL, USA)

1.7.1 About

This is just a minimalistic run through for an example R auto-graded question in Prairie Learn. The question I explain has both auto-graded and manually graded elements. The QID is **HW8_SP2020_part1_autograde_code**.

1.7.2 Directory Structure

```
HW8_SP2020_part1_autograde_code
    info.json
    part1_in.R
    part1_obs_in.R
    question.html

    tests
        part1.R
```

(continues on next page)

(continued from previous page)

```
    │   points.json
    │
    └──tests
            test_00.R
            test_01.R
            test_02.R
            test_03.R
            test_04.R
            test_05.R
            test_06.R
            test_07.R
            test_08.R
            test_09.R
            test_10.R
            test_11.R
```

1.7.3 Explaining the files

`info.json`

```
{
    "uuid": "09b1ad17-f022-4189-b5ce-250743b8f969",
    "title": "Exercise 1: Drawing random numbers-1",
    "topic": "Basic R simulation",
    "tags": ["SP20","easy","Sotiria","code"],
    "type": "v3",
    "singleVariant": true,
    "gradingMethod": "External",
    "externalGradingOptions": {
        "enabled": true,
        "image": "stat430/pl",
        "serverFilesCourse": ["r_autograder/"],
        "entrypoint": "/grade/serverFilesCourse/r_autograder/run.sh",
        "timeout": 60
    }
}
```

If you are coding a new problem while using the same autograder, the things to change would be the `uuid`, `title`, `topic`, `tags`, and `timeout` under the `externalGradingOptions`. The `timeout` is the time in seconds that is allowed for each student submission to be processed. Submission is considered incorrect if it runs longer than the `timeout` duration. Try to keep it minimum (typically 5 to 10 seconds for a small problem, simulations take longer).

`question.html`

```
<div class="card my-2">
  <div class="card-header">
    <b>Exercise</b>
  </div>

  <div class="card-body">

    <pl-question-panel>
          <p>
            Set the seed equal to $61820$. Generate $5, 50, 500, 5,000, 50,000, 5,000,
↪000$ numbers from a $\text{Binomial distribution with } (n=100, p=0.4)$ and assign␣
↪them to variables named $b1, b2, b3, b4, b5, b6$ respectively. Also generate the␣
↪same amount of numbers from a $\text{Poisson distribution with } (λ=40)$ and assign␣
↪them to variables named $p1, p2, p3, p4, p5, p6$. Plot the outputs from each␣
↪experiment in histograms, using function <code>hist()</code>
```

(continues on next page)

(continued from previous page)

```
          </p>
          <p>You may use <code>par(mfrow=c(2,6))</code>, just before the <code>hist()
↪</code> functions, to organize your graphs in $2$ rows of $6$ plots for easier␣
↪comparison. This will result in one row for the Binomial histograms and one row for␣
↪the corresponding (in terms of number of random numbers generated) Poisson␣
↪histograms.</p>
          <pl-file-editor file-name="part1_stu.R" ace-mode="ace/mode/r" source-file-
↪name="part1_in.R"></pl-file-editor>
                </pl-question-panel>
        <pl-submission-panel>
          <pl-file-preview></pl-file-preview>
          <pl-external-grader-results></pl-external-grader-results>
        </pl-submission-panel>
  </div>
</div>

<div class="card my-2">
  <div class="card-header">
    <b>Observation</b>
  </div>

  <div class="card-body">

    <pl-question-panel>
          <p>
            What did you observe from the above experiments? Write as a comment (as R␣
↪comment) in the following window.
          </p>
    <pl-file-editor file-name="part1_obs.R" ace-mode="ace/mode/python" source-file-
↪name="part1_obs_in.R" min-lines="3" auto-resize="true">
    </pl-file-editor>
  </div>
</div>

<div class="card my-2">
  <div class="card-header">
    <b>Plot</b>
  </div>

  <div class="card-body">

    <pl-question-panel>
          <p>
            Upload a PDF of the plot generated from your code. Using the following␣
↪link. Name of the PDF file must be <b>part1_plots.pdf</b>. (Once you have created␣
↪the plots in R, look just above the plots to see and click the Export tab, which␣
↪has the option to export the plots to a .pdf file).
          </p>
          <pl-file-upload file-names="part1_plots.pdf"></pl-file-upload>

        <pl-submission-panel>
          <pl-file-preview></pl-file-preview>
          <pl-external-grader-results></pl-external-grader-results>
        </pl-submission-panel>
  </div>
</div>
```

This is a three part questions, the first `card` shows the autograded portion. The second `card` is the manually graded comment, and the third `card` is the manually graded `.pdf` plot file.

Specifically in the code snippet

```
        <pl-file-editor file-name="part1_stu.R" ace-mode="ace/mode/r" source-file-
↪name="part1_in.R"></pl-file-editor>
```

the `file-name` variable is the what the student submission will be saved as, whereas `source-file-name`

is the starter code which the students will see, and that we need to provide.

`part1_in.R`

```
# Enter code below
```

`part1_obs_in.R`

```
# Enter comment
```

These are the starter code the students see in the code input window for each `card` respectively.

Directory `test`

This directory is only relevant to the autograded portion of the question.

First the two files:

`part1.R`

```
set.seed(61820)
b1 = rbinom(5,size=100,prob=0.4)
b2 = rbinom(50,size=100,prob=0.4)
b3 = rbinom(500,size=100,prob=0.4)
b4 = rbinom(5000,size=100,prob=0.4)
b5 = rbinom(50000,size=100,prob=0.4)
b6 = rbinom(5000000,size=100,prob=0.4)
p1 = rpois(5,40)
p2 = rpois(50,40)
p3 = rpois(500,40)
p4 = rpois(5000,40)
p5 = rpois(50000,40)
p6 = rpois(5000000,40)
par(mfrow=c(2,6))
hist(b1)
hist(b2)
hist(b3)
hist(b4)
hist(b5)
hist(b6)
hist(p1)
hist(p2)
hist(p3)
hist(p4)
hist(p5)
hist(p6)
```

This file contains the code, which is the correct solution of the problem.

`points.json`

```
[
    {
        "name":"Test b1",
        "file":"test_00.R",
        "max_points":2
    },

    {
        "name":"Test b2",
        "file":"test_01.R",
        "max_points":2
    },
    {
        "name":"Test b3",
        "file":"test_02.R",
        "max_points":2
    },
    {
        "name":"Test b4",
        "file":"test_03.R",
        "max_points":2
    },
    {
        "name":"Test b5",
        "file":"test_04.R",
        "max_points":2
    },
    {
        "name":"Test b6",
        "file":"test_05.R",
        "max_points":2
    },
    {
        "name":"Test p1",
        "file":"test_06.R",
        "max_points":2
    },
    {
        "name":"Test p2",
        "file":"test_07.R",
        "max_points":2
    },
    {
        "name":"Test p3",
        "file":"test_08.R",
        "max_points":2
    },
    {
        "name":"Test p4",
        "file":"test_09.R",
        "max_points":2
    },
    {
        "name":"Test p5",
        "file":"test_10.R",
        "max_points":2
    },
```

(continues on next page)

(continued from previous page)

```
    {
        "name":"Test p6",
        "file":"test_11.R",
        "max_points":2
    }
        ]
```

This file list the name of the unit tests (make them relevant to what you are testing as the student will see which tests the student passed or not and modify their submission accordingly), the name of the file for all the unit test, and the points for passing the unit tests.

The unit tests themselves are in another subdirectory named tests, lets call it the nested directory `tests`.

Nested Directory `tests`

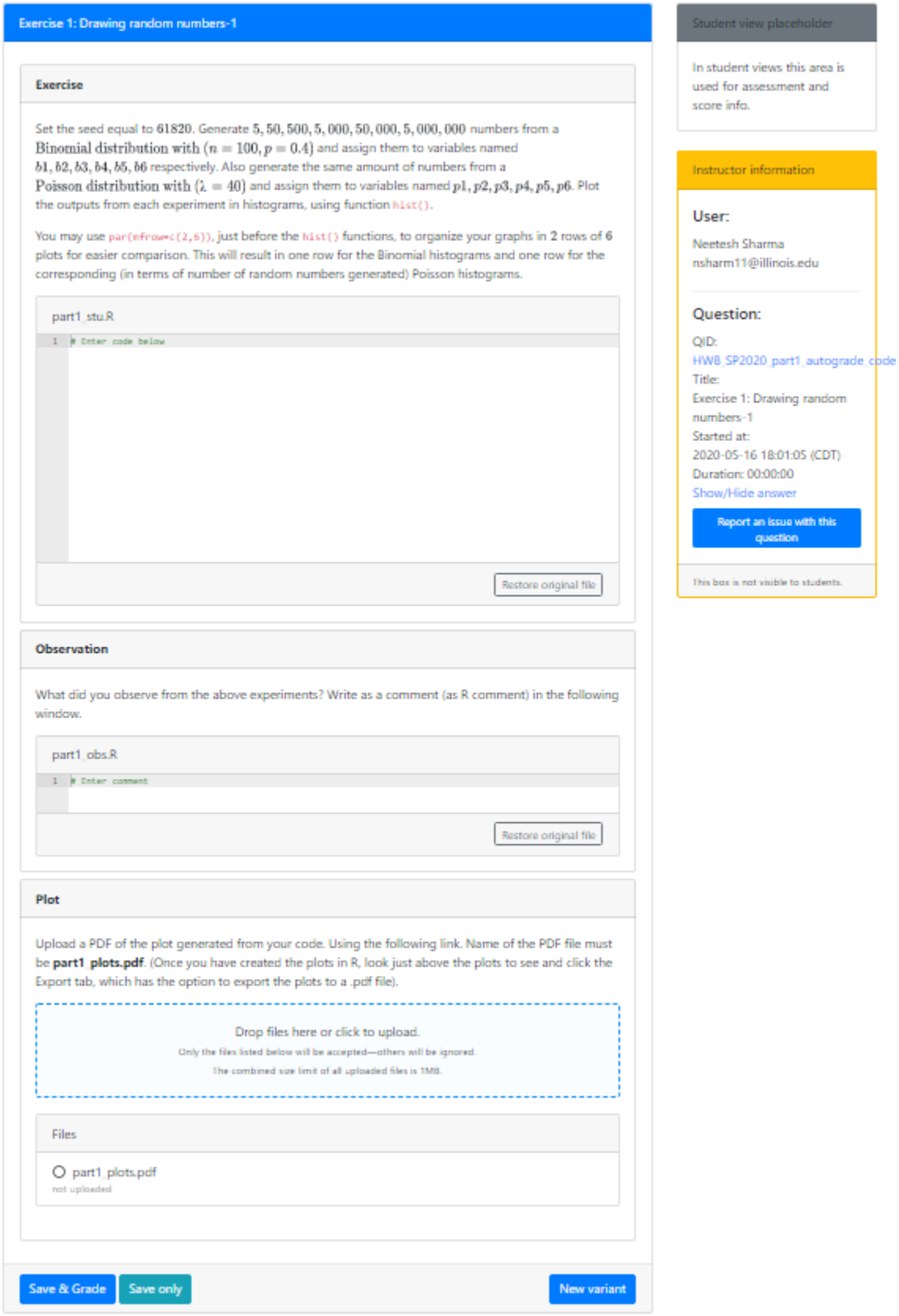
I will just explain one of the tests

```
# load student results
Sys.chmod("/grade/student/part1_stu.R", mode="0664")
student <- unix::eval_safe({source("/grade/student/part1_stu.R"); b1}, uid=1001)

# load correct results
source("/grade/tests/part1.R")
correct <- b1

#compare
using(ttdo)
expect_equivalent_with_diff(student, correct, mode="unified", format="ansi256")
```

The unit tests have a simple structure with three steps:

1. Load the student results by running the submitted code, and extracting any variable or function evaluation of interest. The complicated way of running the student code is due to security considerations.
2. Run the correct source code and extract the corresponding benchmark result.
3. Compare the two

Finally the questions renders as follows


Exercise 1: Drawing random numbers-1
Exercise
Set the seed equal to 61820. Generate 5, 50, 500, 5, 000, 50, 000, 5, 000, 000 numbers from a Binomial distribution with (n = 100, p = 0.4) and assign them to variables named b1, b2, b3, b4, b5, b6 respectively. Also generate the same amount of numbers from a Poisson distribution with (λ = 40) and assign them to variables named p1, p2, p3, p4, p5, p6. Plot the outputs from each experiment in histograms, using function hist().
You may use par(mfrow=c(2,6)), just before the hist() functions, to organize your graphs in 2 rows of 6 plots for easier comparison. This will result in one row for the Binomial histograms and one row for the corresponding (in terms of number of random numbers generated) Poisson histograms.
part1_stu.R
Enter code below
Restore original file
Observation
What did you observe from the above experiments? Write as a comment (as R comment) in the following window.
part1_obs.R
Enter comment
Restore original file
Plot
Upload a PDF of the plot generated from your code. Using the following link. Name of the PDF file must be part1_plots.pdf. (Once you have created the plots in R, look just above the plots to see and click the Export tab, which has the option to export the plots to a .pdf file).
Drop files here or click to upload.
Only the files listed below will be accepted—others will be ignored.
The combined size limit of all uploaded files is 1MB.
Files
part1_plots.pdf
not uploaded
Save & Grade
Save only
New variant
Student view placeholder
In student views this area is used for assessment and score info.
Instructor information
User:
Neetesh Sharma
nsharm11@illinois.edu
Question:
QID:
HW8_SP2020_part1_autograde_code
Title:
Exercise 1: Drawing random numbers-1
Started at:
2020-05-16 18:01:05 (CDT)
Duration: 00:00:00
Show/Hide answer
Report an issue with this question
This box is not visible to students.

1.7.4 Closing statement

This example does not follow all the recommended guidelines, for example it is recommended that the student code submission be a function and not a script. However, CEE202 being a beginner course the students are expected to only work with basic scripting. Maybe the question can be improved in future if we wrap the student code in the back end to be run as a function. Furthermore, I would recommend not having different grading methods in the same question, as it confuses the students on the total marks they got, as the manually graded parts are uploaded separately. However, this was among the messiest questions we had so it was a good example to explain various possibilities. Thanks!

Useful link: An R Autograder for PrarieLearn

1.8 Review of autograding example R notebook

By Advai Podduturi (Department of CS, University of Illinois, Urbana-Champaign, IL, USA)

1.8.1 About

This guide will cover how to write a deploy an R notebook as an autogradable assignment on Prairielearn.

1.8.2 Directory Structures

```
WSXX_Example_Topic
|   info.json
|   question.html
|
|___tests
|   points.json
|   |   ans1.R
|   |   ans2.R
|   |   ans3.R
|   |___tests
|   |   |   test1.R
|   |   |   test2.R
|   |   |   test3.R
|   |
|___workspace
|   | Workbook.ipynb
|
```

1.8.3 Explaining the files

`info.json`

```
{
    "uuid": "09b1ad17-f022-4189-b5ce-250743b8f969",
    "title": "WSXX Example Topic",
    "topic": "Basic R Notebook",
    "tags": [
        "Spring2022",
        "Sotiria",
        "Priyam",
```

(continues on next page)

(continued from previous page)

```
        "CLT",
        "Jupyter"
    ],
    "type": "v3",
    "singleVariant": true,
    "workspaceOptions": {
        "image": "prairielearn/workspace-jupyterlab",
        "port": 8080,
        "home": "/home/jovyan",
        "rewriteUrl": false,
        "gradedFiles": [
            "Workbook.ipynb"
        ]
    },
    "gradingMethod": "External",
    "externalGradingOptions": {
        "enabled": true,
        "image": "advai/grader-r-advai",
        "serverFilesCourse": [
            "r_autograder/"
        ],
        "entrypoint": "/grade/serverFilesCourse/r_autograder/run.sh",
        "timeout": 20
    }
}
```

If you are coding a new problem while using the same autograder, the things to change would be the `uuid`, `title`, `topic`, `tags`, and `timeout` under the `externalGradingOptions`. The `timeout` is the time in seconds that is allowed for each student submission to be processed. Submission is considered incorrect if it runs longer than the `timeout` duration. Try to keep it minimum (typically 5 to 10 seconds for a small problem, simulations take longer).

The R notebook autograder image is stored under advai/grader-r-advai.

`question.html`

I typically use the same template for this that just displays the button for opening the Prairielearn Jupyter interface.

```
<pl-question-panel>
  <p>This is a workspace question with an in-browser <a href="https://jupyter.org">
↪JupyterLab</a>.
  </p>

  <p>
      In this worksheet, you will be learning about and understanding "Example Topic".
  </p>

  <p>
      Once in the JupyterLab environment, please, open workbook called <b>Workbook.
↪ipynb</b>. After you complete your code in it, save the workbook and come back to␣
↪this Prairie Learn question window to click Save and Grade button. Ignore the other␣
↪.ipynb workspace you see. </p>

  <pl-external-grader-variables params-name="names_from_user"></pl-external-grader-
↪variables>

  <pl-workspace></pl-workspace>
</pl-question-panel>
```

(continues on next page)

(continued from previous page)

```
<pl-submission-panel>
  <pl-external-grader-results></pl-external-grader-results>
  <pl-file-preview></pl-file-preview>
</pl-submission-panel>
```

1.8.4 Subdirectory: `tests`

`points.json`

You can define any number of tests you want by just creating more test files under tests/tests. To assign points to these tests, you need to edit the points.json.

```
[
    {
        "name":"Test 1",
        "file":"test1.R",
        "max_points":1
    },

    {
        "name":"Test 2",
        "file":"test2.R",
        "max_points":3
    },

    {
        "name":"Test 3",
        "file":"test3.R",
        "max_points":5
    }
]
```

`ans1.R`

```
#grade {p1_t0, p1_ta, p1_pvalue}
#1) Fixed probability track
#calculate the t-statistic from the data
p1_alpha<- 0.05 # Alpha Value
p1_xbar <- 1.9 # Sample Mean of 25 trips
p1_mu<- 2.0 # Traditional fuel intensity
p1_s<- 0.2 # Standard deviaion
p1_n<- 25 # Sample size
p1_t0 <- round((p1_xbar - p1_mu)/ (p1_s/p1_n^0.5), 3) # critical value of null

#find the t-value for the significance level 0.05 with n-1 dof
p1_ta <- round(qt(p1_alpha, (p1_n-1), lower.tail=TRUE, log.p=FALSE), 3) # critical␣
↪value of alternative

p1_pvalue <- round(pt(p1_t0, (p1_n-1), lower.tail=TRUE, log.p=FALSE), 3) # P value
```

Here we can see that all the supporting variables for the p1_t0, p1_ta, and p1_pvalue computations are available in the ans1.R file. Not including the right variables is a common way to make bugs when autograding R notebooks. **Use unique variable names across testX/ansX.R files or they will clash and cause one of the tests to fail.**

`tests/test1.R` I will only go over one test for brevity but consult the tests folder of any question in the PL-CEE202 question bank for more examples of tests. I will highlight three important notes

1. All of the student's code from the notebook gets written to `/grade/student/stcode.R` so be sure to source from that location.
2. Be sure to add #grade tags to all essential cells so that all the important supporting data is written to `grade/student/stcode.R`.
3. Use unique variable names across tests or they will clash. A common approach is "pX_variable".

You test code by sourcing the student's value for a variable and then sourcing the correct value from the `ansX.R` file. You can compare them using expect_equivalent_with_diff() as shown below.

```
Sys.chmod("/grade/student/stcode.R", mode="0664")
student_p1_t0 <- unix::eval_safe({source("/grade/student/stcode.R"); p1_t0}, uid=1001)
student_p1_ta <- unix::eval_safe({source("/grade/student/stcode.R"); p1_ta}, uid=1001)
student_p1_pvalue <- unix::eval_safe({source("/grade/student/stcode.R"); p1_pvalue},␣
↪uid=1001)

source("/grade/tests/ans1.R")
correct_p1_t0 <- p1_t0
correct_p1_ta <- p1_ta
correct_p1_pvalue <- p1_pvalue

using(ttdo)
expect_equivalent_with_diff(student_p1_t0, correct_p1_t0, mode="unified", format=
↪"ansi256")
expect_equivalent_with_diff(student_p1_ta, correct_p1_ta, mode="unified", format=
↪"ansi256")
expect_equivalent_with_diff(student_p1_pvalue, correct_p1_pvalue, mode="unified",␣
↪format="ansi256")
```

1.8.5 Subdirectory `workspace`

`Workbook.ipynb`

It is imperative that the notebook be named "Workbook.ipynb" or the autograder will not pick it up. Typically, notebooks are designed as lesson plans. To grade a cell in the notebook, simply add

```
#grade
```

to the top of the cell. Note: if a problem uses variables across multiple cells, then you need #grade tags in all those cells.

1.9 Testing gradability of your R Notebook

The intial set up is the hardest/longest part. After that, developing in a local PL env is very easy.

1.9.1 Setting up local development environment

1. Install docker
2. Clone the PL CEE Repo Now you should be able to launch a local PL instance by running

```
./runMe.sh
```

1.9.2 Adding notebook to course directory

Create a new folder in `questions/` for your worksheet and fill out all the relevant content descirbed above. Then you can navigate to http://localhost:3000/ to view your question.

You should also create a copy of the folder and add `_filled` to the end of the folder name. This is where you safely store your filled notebook without risk of leaking it to students. **Make sure to change the uuid in the `info.json` or it will clash with the original problem in PL.**

1.9.3 Testing

It's important to test that each individual question grades since students will work incrementally. Launch the question and replace each cell block with the filled cell block and grade to ensure the question is being graded properly.

1.9.4 Deploying to live PL

1.10 Final Thoughts

`questions/WS13_Central_Limit_Theorem` is a great example question to reference.

1.11 List of Topics and Tags

1.11.1 Topics

Reference: (link)

Each question in the course has a topic from the list specified in the `infoCourse.jsonfile`. Topics should be thought of as **chapters** or **sections** in a textbook, and there should be about 10 to 30 topics in a typical course. The topic properties are as follows.

For example, topics could be listed like:

```
"topics": [
  {"name": "Vectors", "color": "blue3", "description": "Vector algebra in 3D."},
  {"name": "Center of mass", "color": "green3", "description": "Calculating the␣
↪center of mass for irregular bodies and systems."}
],
```

1.11.2 Tags

Reference: (link)

1.12 Grade Transfer from PL to Compass

Provided by Sophia Hoang:

1. download the gradebook csv files from both PrairieLearn (PL) and Compass2g (compass)
2. sort the compass gradebook alphabetically by net id since PL is sorted by netid
3. make sure the names on the PL gradebook match up with the compass gradebook
4. copy and paste the column of numbers from PL to compass
5. upload the updated file back onto compass

1.13 Attendance and muddiest point compilation

By Neetesh Sharma

1.13.1 Summary

This is simple python script, which is written to compile data collected by responses from the team based class attendance form and the muddiest point question. The script works with these specific questions and the way prairie learn output csv files look like. The code is hacky and not particularly clean or efficient but anyone with basic idea of file manipulation and scripting in python should be able to read and edit to make it work with different problems.

1.13.2 Description

In this section I will just go through the different parts of the script and give brief descriptions.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 28 14:41:03 2020

@author: nsharm11@illinois.edu
"""
import pandas as pd
import numpy as np
import glob
import Levenshtein as lv
```

Importing the libraries.

1. `Pandas` for manipulating data in dataframes and input output of csv.
2. `Numpy` for some basic array functions
3. `Glob` to collect files from the student file submission directory
4. `Levenshtein` to calculate the string similarity with input student names and student names in roster to detect and correct spelling mistakes by students in filling their names

```
############################
###### Inputs #############
############################
rosterpath = "roster3-25-20.csv"
```

(continues on next page)

(continued from previous page)

```
filespath = "Worksheet22/*.txt"
scorepath = "CEE_202_SPRING2020_WS22_scores_by_username.csv"
max_score = 0
att_outfile = "Attendance_WS_compiled_WS22.csv"
mp_outfile = "Muddiest_point_compiled_WS22.csv"
date = "apr30"
```

Now we move on to the inputs.

1. `roster3-25-20.csv` is the class roster with the following format:

```
team,last,first,netid
Team_1,Bellary,Amritha,abella8
Team_1,Barbieri,Giulia,gbarbier
Team_1,Osei,Kweku,kosei2
Team_1,Wiggins,Robert,rjw5
Team_2,Nguyen,Chris,cnguye52
Team_2,Ambrosino,Jack,jackaa2
Team_2,Salam,Shaadmaan,sfsalam2
```

2. filespath is where the "best_files" from PL are located, as obtained from the downloads available on PL
3. scorepath is again a csv of scores by username as downloaded from PL
4. maxscore is the maximum score in the assessment from the above file (Note that in the current class policy `>=50%` of the max score earns `100` points and zero otherwise)
5. Then there are the preferred names of the outputs and the date of the worksheet being processed

```
# Read the roster and make all possible mapping dicts for convenience

roster=pd.read_csv(rosterpath)
roster['name']=(roster['last']+roster['first'])
roster['name1']=(roster['first']+roster['last'])
roster['name'] = [''.join(filter(str.isalnum, name)).lower().strip().replace(" ","")␣
↪for name in roster['name']]
roster['name1'] = [''.join(filter(str.isalnum, name)).lower().strip().replace(" ","")␣
↪for name in roster['name1']]
roster['att'] = 0
roster['score'] = 0
roster['submitting_id'] = ''
roster['check'] = 0

nametoid = {}
idtoname = {}
idtoteam = {}
for i in roster.index:
    nametoid[roster.loc[i,'name']]= roster.loc[i,'netid']
    nametoid[roster.loc[i,'name1']]= roster.loc[i,'netid']
    idtoname[roster.loc[i,'netid']]= roster.loc[i,'name']
    idtoteam[roster.loc[i,'netid']]= roster.loc[i,'team']

teams = roster.groupby('team').groups
teamtonames = {}
teamtoids = {}

for key in teams.keys():
    teamtonames[key] = roster.loc[teams[key],['name','name1']].values.flatten()
```

(continues on next page)

(continued from previous page)

```
    teamtoids[key] = roster.loc[teams[key],'netid'].values
roster.index = roster.netid
```

We then process the roster and make mappings from id to name and name to id, id to team and team to multiple ids. These will be useful for processing the student inputs. Also, the processed `roster` dataframe serves as place to report the attendance and scores, and that is why I add numeric columns for `'att', 'score', 'submitting_id'`, and `'check'` in the roster as place holders.

```
# Read submitted files and separate present team member names

flist = glob.glob(filespath)
df1 = pd.DataFrame([chunks(filespath,f) for f in flist])

qgrp=df1.groupby(['question']).groups
dfnames = df1.loc[qgrp['team_names.txt'],:]
dfnames.index = dfnames.netid
dfnames.sort_values(['n2','n3'],ascending=[0,0],inplace=True)
```

We then read the submitted files and and extract data from the filename as well as the text inside the files. The function `chunks` performs this procedure for each filename

```
def chunks(filespath,fname):
    allchunks = fname.split('\\')[-1]
    allchunks = allchunks.split('_')
    semail = allchunks[0]
    ## CHange teh number to extract netid
    netid = semail.split('@')[0]
    if allchunks[5]=='File':
        qname = allchunks[-2]+'_'+allchunks[-1]
        n1 = int(allchunks[1])
        n2 = int(allchunks[7])
        n3 = int(allchunks[8])
    else:
        qname = allchunks[-2]+'_'+allchunks[-1]
        n1 = int(allchunks[1])
        n2 = int(allchunks[7])
        n3 = int(allchunks[8])
    with open(fname,"r") as file1:
        ftxt = file1.read()
    return {
            'netid':netid,
            'question':qname,
            'n1':n1,
            'n2':n2,
            'n3':n3,
            'ftext': np.array(''.join(filter(str.isalnum, ftxt.strip().replace('\n',
↪'zzz').replace('\r', 'zzz').replace(', ', '').replace(' ', '').lower())).split('zzz
↪')),
            'ftext_ue': ftxt}
```

This function is the piece which would need editing if the code is to be applied to a different problem, since `chunks` relies on the position of different type of information at different location inside the filename.

```
# Read scores
score = pd.read_csv(scorepath)
score.columns = ['netid','raw']
```

(continues on next page)

(continued from previous page)

```
score['score'] = 100*(score['raw']>=max_score/2)
score.index = score.netid
score = score.score
```

We then read the score file as well and create a id to score map.

```
checkdf = pd.DataFrame([],columns=['Typo', 'netid', 'Correct'])
for netid in dfnames['netid']:
    roster.loc[netid,'att']=1
    roster.loc[netid,'submitting_id']=netid
    roster.loc[netid,'score'] = max(roster.loc[netid,'score'],score[netid])
    names = dfnames.loc[netid,'ftext']
    if type(names) !=type(np.array([])):
        names = names.values[0]
    for name in names:
        try:
            roster.loc[nametoid[name],'att']=1
            roster.loc[nametoid[name],'submitting_id']=netid
            roster.loc[nametoid[name],'score'] = max(roster.loc[nametoid[name],'score
↪'],score[netid])
        except:
            team = idtoteam[netid]
            candidate_names = teamtonames[team]
            foundflag = False
            for cn in candidate_names:
                if lv.distance(cn,name) < 7:
                    foundflag = True
                    roster.loc[nametoid[cn],'att']=1
                    roster.loc[nametoid[cn],'submitting_id']=netid
                    roster.loc[nametoid[cn],'score']=max(roster.loc[nametoid[cn],
↪'score'],score[netid])
                    roster.loc[nametoid[cn],'check']=1
                    checkdf=checkdf.append({'Typo':name, 'netid':netid, 'Correct':cn},
↪ignore_index=True)
                else:
                    continue
            if foundflag == False:
                checkdf=checkdf.append({'Typo':name, 'netid':netid, 'Correct':'Not␣
↪found'},ignore_index=True)
```

This part of code now gives the attendance to the students listed inside the present team members portion. We first give attendance to the submitting id, we then try to map name to ids using the roster data and if no name to id is found we try to compare with the available ones using the string comparison and try to make corrections. If no correction is found within the search distance, `"not found"` is reported. We keep track of all the correction we made in a `checkdf` dataframe.

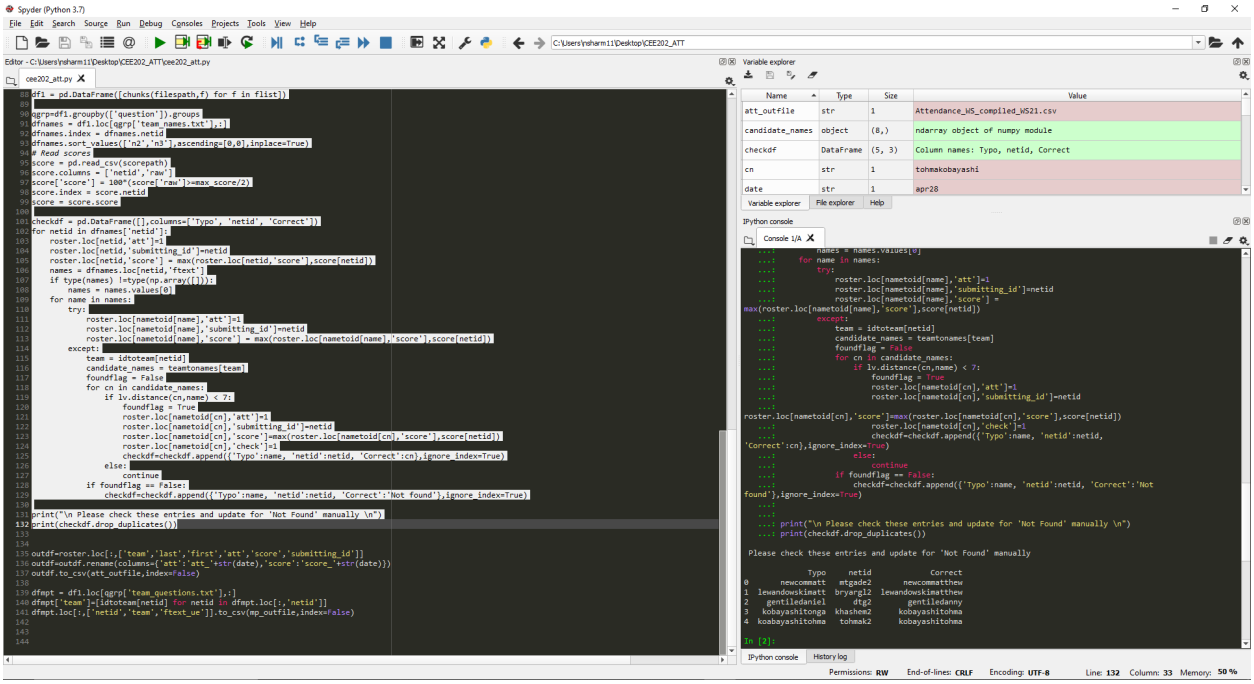
```
print("\n Please check these entries and update for 'Not Found' manually \n")
print(checkdf.drop_duplicates())


outdf=roster.loc[:,['team','last','first','att','score','submitting_id']]
outdf=outdf.rename(columns={'att':'att_'+str(date),'score':'score_'+str(date)})
outdf.to_csv(att_outfile,index=False)

dfmpt = df1.loc[qgrp['team_questions.txt'],:]
dfmpt['team']=[idtoteam[netid] for netid in dfmpt.loc[:,'netid']]
dfmpt.loc[:,['netid','team','ftext_ue']].to_csv(mp_outfile,index=False)
```

We then print the corrections we made for a manual check, and then write the output files.

1.13.3 Running the code

Use a GUI such as spyder to run the script. Make sure you are in the relevant working directory and files are in place according to the paths you define in the inputs.

Fill in all the inputs and run the code upto the print out of the `checkdf`.



Carefully check the printed corrections

```
 Please check these entries and update for 'Not Found' manually

             Typo     netid           Correct
0       newcommatt   mtgade2       newcommatthew
1  lewandowskimatt  bryargl2  lewandowskimatthew
2    gentiledaniel      dtg2        gentiledanny
3   kobayashitonga  khashem2      kobayashitohma
4  koabayashitohma   tohmak2      kobayashitohma

In [2]:
```

If there are some not founds or something the script got wrong, we need to edit the files submitted by the identified student id manually.

Finally run the code for outputs and processing is done.

1.14 Markdown Introduction

1.14.1 What is Markdown?

According to wiki:

> **Markdown** is a lightweight markup language with plain-text-formatting syntax. Its design allows it to be converted to many output formats, but the original tool by the same name only supports HTML. Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor.

1.14.2 How to open and create the Markdown files?

- I recommend using the markdown editor Typora, which gives you a seamless experience as both a reader and a writer. It is free fo Windows/MacOS/Linux users.
- But you can use any text editor (such as Notepad++ on **Windows** or Sublime Text on **MacOS/Linux**) to open and create the `*.md` files.

1.14.3 How to write?

Important: **`Heading 1`** of your markdown file (**not your markdown file name**) will be the section name showing on the webpage.

To create a markdown-based documentation:

- You are not required to learn any syntax if you are using Typora.
- You can learn basic Markdown syntax within 5 mins. See the resources:
  - **Markdown cheatsheet from Markdown Guide** (link)
  - **Markdown cheatsheet from GitHub** (link)
  - **Extended syntax** (link)

1.15 Maintain Website

1.15.1 Prerequisites

- Fork or Clone the GitHub from `https://github.com/zzheng93/pl-cee202-docs`
- Have the markdown file `*.md` ready. This file will be a page for this website

1.15.2 Step 1: Include your markdown file

- Include your markdown file in the folder `pl-cee202-docs/source/page/`

  Note: if your Markdown file name is `intro.md`, and you want this page to be under the section `WEBSITE DEVELOPERS`, then the directory of this file is `pl-cee202-docs/source/page/web/intro.md`.

1.15.3 Step 2: Update the index.rst

Important: **`Heading 1`** of your markdown file (**not your markdown file name**) will be the section name showing on the webpage.

- Use text editor (such as Notepad++ on **Windows** or Sublime Text on **MacOS/Linux**) to open the `index.rst` (by the directory `pl-cee202-docs/source/index.rst`)
- Find the following contents (note: the name will be slightly different).

```
Contents
--------
.. toctree::
   :maxdepth: 2
   :caption: Students

   page/student/<*.md>

.. toctree::
   :maxdepth: 2
   :caption: Instructors/TAs

   page/instructor_TA/<*.md>

.. toctree::
   :maxdepth: 2
   :caption: Website Developers

   page/web/markdown_intro.md
   page/web/maintain_site.md
```

- If you want to add your page under the "Website Develops" section in the left panel, add the name page/web/intro.md under the corresponding section :caption: Website Developers. Note the difference. Here you don't need to include the path prefix "pl-cee202-docs/source/", because you only need to define the relative path.

```
Contents
--------
.. toctree::
   :maxdepth: 2
   :caption: Students

   page/student/<*.md>

.. toctree::
   :maxdepth: 2
   :caption: Instructors/TAs

   page/instructor_TA/<*.md>

.. toctree::
   :maxdepth: 2
   :caption: Website Developers

   page/web/markdown_intro.md
   page/web/maintain_site.md
   page/web/intro.md
```

1.15.4 Step 3: Commit and push to GitHub

- If you have the access permission to this GitHub, you can commit push.
- If you don't have the permission, you can always create a pull request (how to creat a pull request?).

CHAPTER 2

How to ask for help

The GitHub issue tracker is the primary place for bug reports.