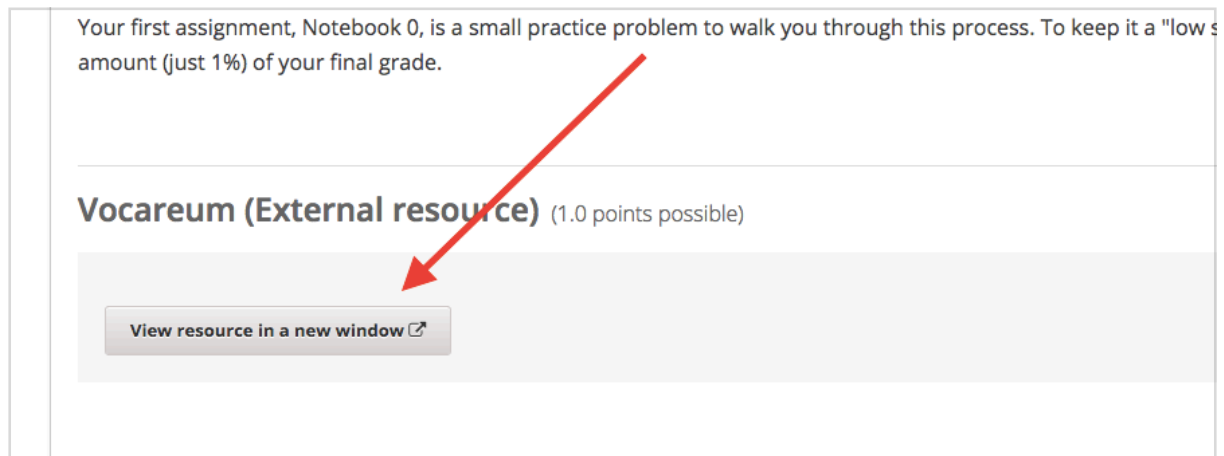


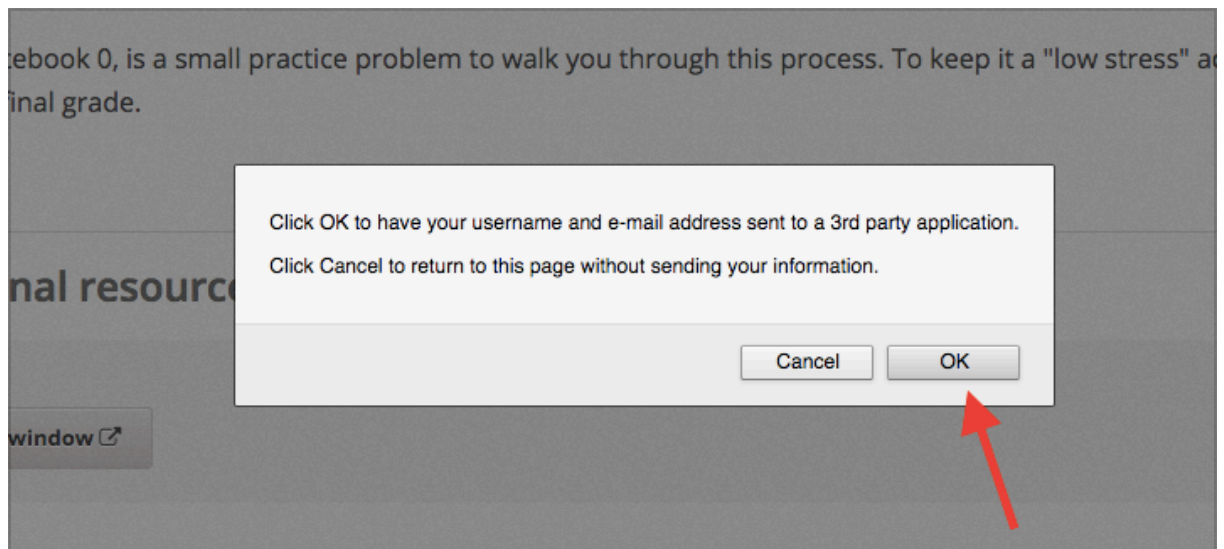
CSE 6040/x: How to submit Notebook 0—a tutorial on Jupyter and Vocareum

Launching Vocareum

To launch Vocareum, start by clicking the link, `View resource in a new window`:

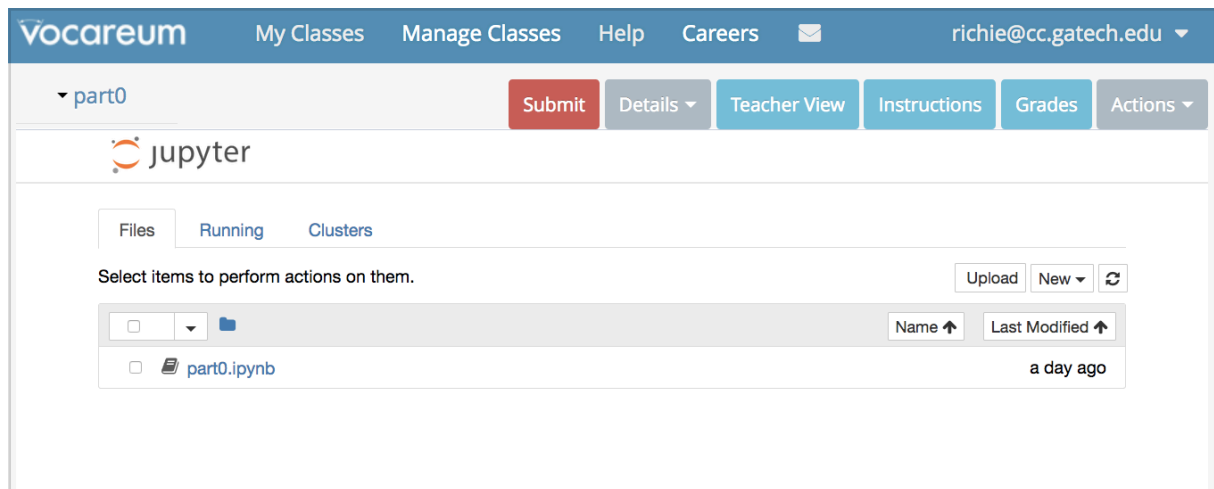


When prompted, accept the request to have your username and email sent from edX to Vocareum. This step is necessary to give you credit when you eventually submit the assignment on Vocareum.

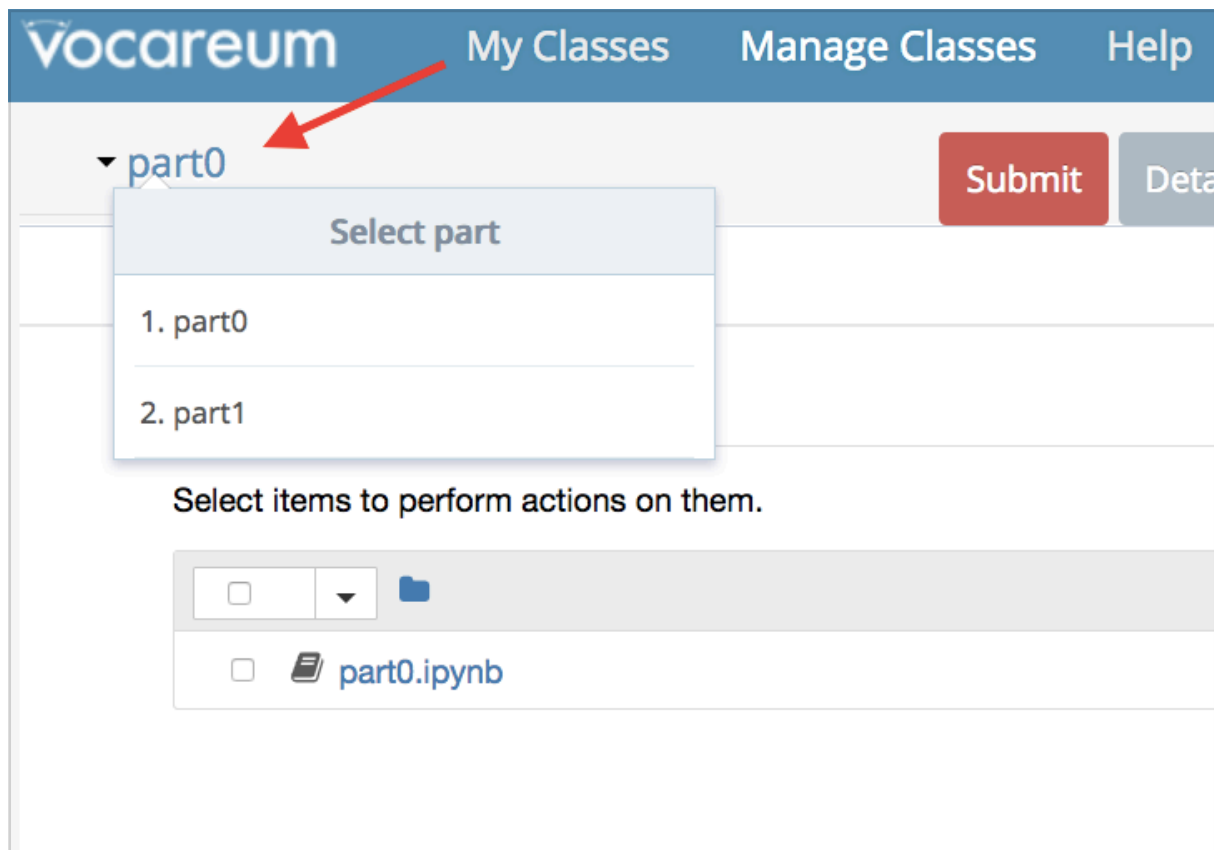


Jupyter

When you "arrive" on the Vocareum side, you should see a Jupyter file browser running. Take note of the elements highlighted below:



In the upper-left corner, notice the dropdown menu labeled, `part0`. If a notebook assignment has multiple parts, you can switch between those parts here.



Parts of an assignment are treated separately. As such, the usual "workflow" is to submit each part as you complete it. That also means you need to submit **all** parts of a notebook to get full credit.

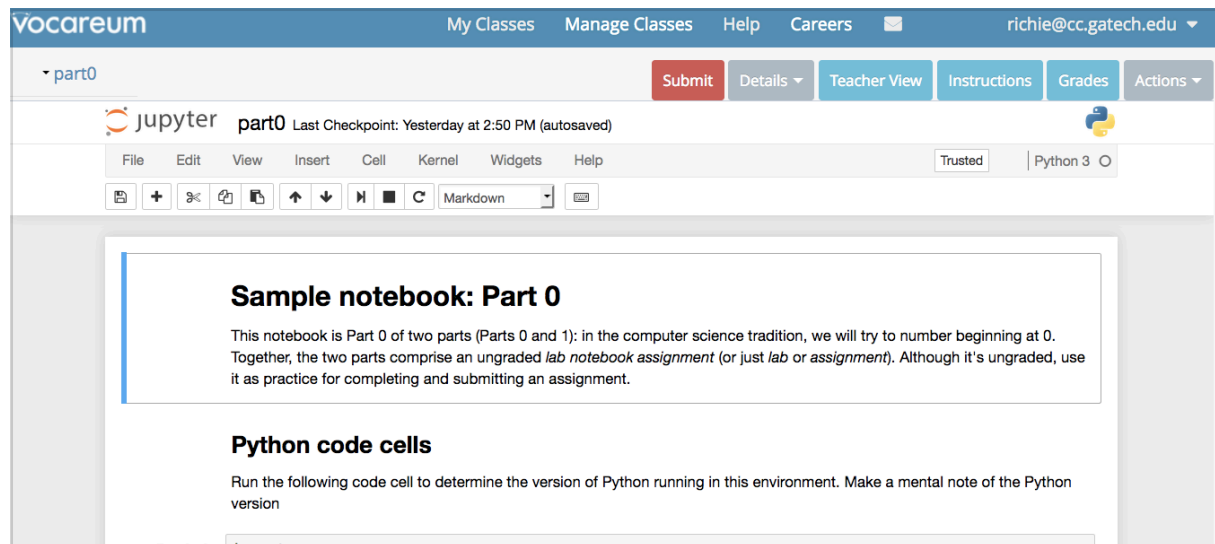
For now, let's focus on `part0`. To open the notebook for `part0`, click on `part0.ipynb`.



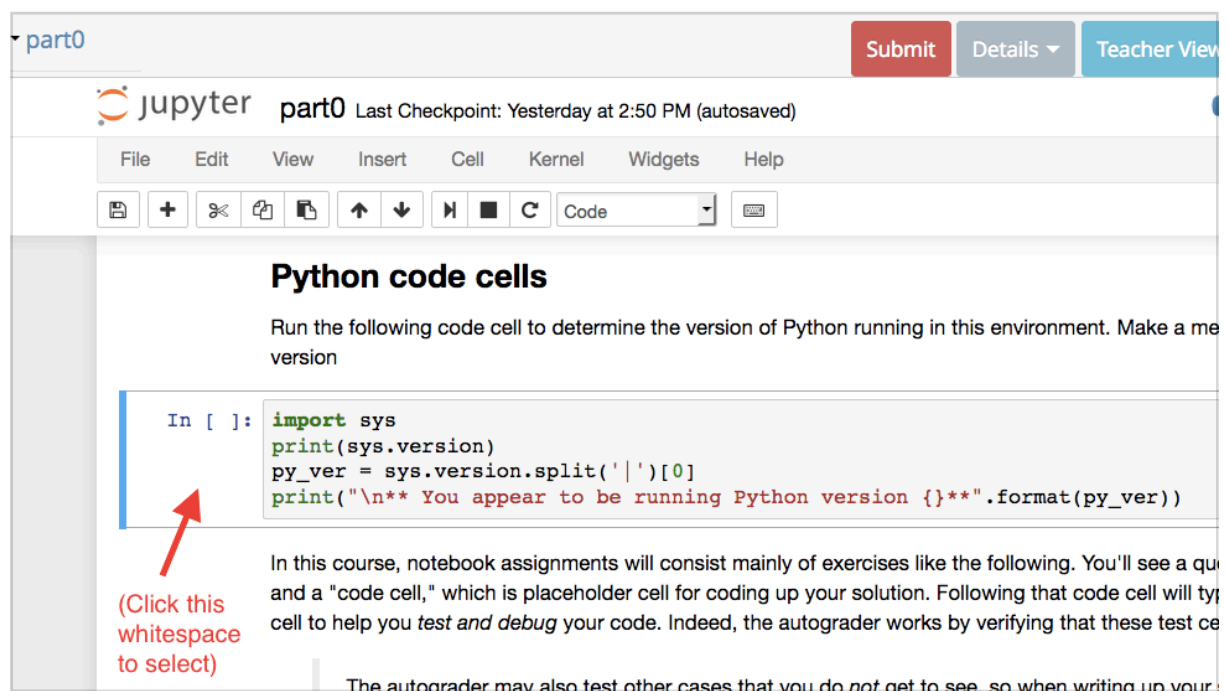
The extension, `.ipynb`, is short for "IPython notebook." This name is an historical artifact, when the notebook system only supported Python. (It now supports many other languages, including the Julia, Python, and R from which "Jupyter" derives!

Working with notebooks

If the above worked, you should see a notebook!



A notebook consists of **cells**. In the image above, the box with the thicker light blue edge highlights the first cell. Cells may contain text, like the highlighted cell above, or code, like the third cell. Take a closer look at that first code cell. (If you need to, scroll the page so that it becomes visible; then, to highlight it, click anywhere in the whitespace immediately to the left of the code cell.)



When that code cell is selected, a status indicator in the toolbar will read `Code`:

part0

Submit Details Teacher View

jupyter part0 Last Checkpoint: Yesterday at 2:50 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

Python code cells

Run the following code cell to determine the version of Python running in this environment. Make a me version

```
In [ ]: import sys
print(sys.version)
py_ver = sys.version.split('|')[0]
print("\n** You appear to be running Python version {}".format(py_ver))
```

In this course, notebook assignments will consist mainly of exercises like the following. You'll see a qu and a "code cell," which is placeholder cell for coding up your solution. Following that code cell will typ cell to help you *test and debug* your code. Indeed, the autograder works by verifying that these test ce

The autoorader mav also test other cases that you do *not* aet to see. so when writing up your:

Indeed, that indicator is actually a dropdown box that you can use to change the type of any cell. In this case, we want this cell to be code, so leave it as is.

Since the cell has code in it, you can run it! In fact, notice the label to the left of the cell, which reads, `In []:`. The blank enclosed in square brackets indicates that this code cell has **not** yet been run. Let's do so, by clicking on the right-facing triangle in the toolbar:

er part0 Last Checkpoint: Yesterday at 2:50 PM (autosaved)

t View Insert Cell Kernel Widgets Help

Code

Python code cells

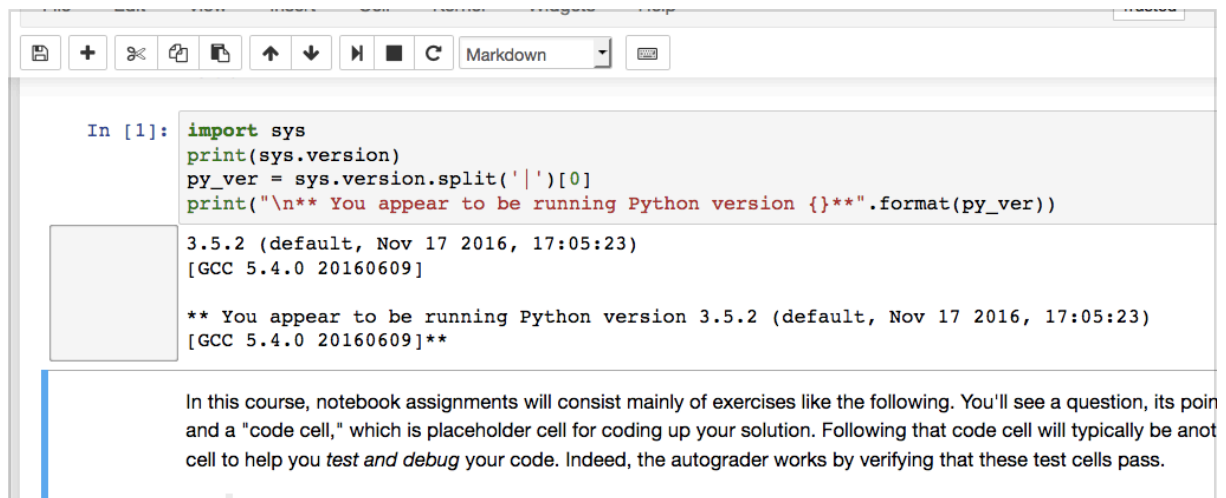
Run the following code cell to determine the version of Pyth version

```
] : import sys
print(sys.version)
```

You can also use a keyboard shortcut to execute the currently selected cell. See the [Help](#)

menu for a list.

The result should look something like the following:



The screenshot shows a Jupyter Notebook window. At the top is a toolbar with icons for saving, adding, deleting, and other actions, along with a 'Markdown' dropdown menu. Below the toolbar is a code cell labeled 'In [1]:'. The code inside the cell is:

```
import sys
print(sys.version)
py_ver = sys.version.split('|')[0]
print("\n** You appear to be running Python version {}".format(py_ver))
```

Below the code cell is the output, which is displayed in a light gray box. The output consists of three lines:

```
3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609]

** You appear to be running Python version 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609]**
```

Below the output is a text cell containing the following text:

In this course, notebook assignments will consist mainly of exercises like the following. You'll see a question, its point value, and a "code cell," which is placeholder cell for coding up your solution. Following that code cell will typically be another cell to help you *test and debug* your code. Indeed, the autograder works by verifying that these test cells pass.

The output of this code appears immediately after the cell. Indeed, this output is now **part** of the document! That's what we mean when we say that a Jupyter notebook is like an executable report: the code you include can be used to generate output that becomes a part of the report.

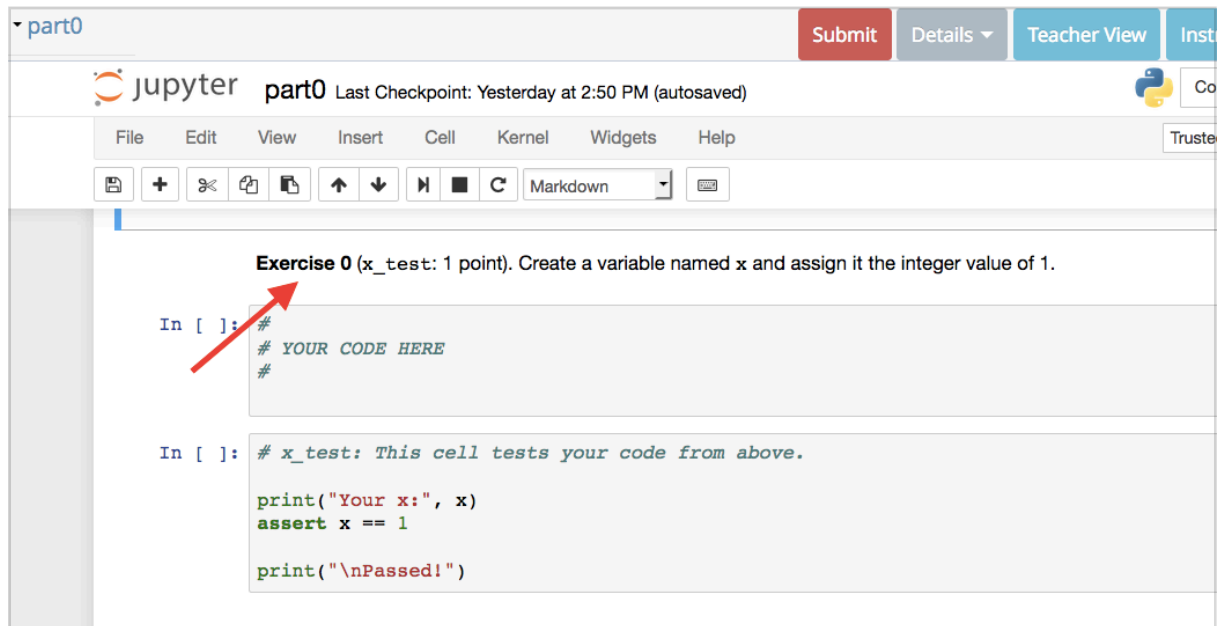
Thought question: In this example, we are printing the version of Python that is being executed, along with a bunch of other information. Why might someone want to do that inside his or her notebook?

Also observe that the cell's label has changed from `In []:`, which meant "has not been executed," to `In [1]:` above. The value (here, `1`) is a **counter** that indicates how many times any code cells (not just the selected one) have been run. In this case, **any** code cell you run next---no matter where it is in the notebook---will get the label, `In [2]:`.

Indeed, as you are working with a notebook, one of the "dangerous freedoms" you have is the ability to move around the document and execute code cells in any order. That can lead to unusual behavior. It's fine to do that as you are developing and debugging a notebook. However, when we (auto)grade your notebooks, we will execute all code cells in sequence from top to bottom, so you need to be sure that the notebook works correctly in that scenario. We will revisit this idea before submitting.

A sample exercise

If you scroll down a bit further, you'll encounter your first exercise, **Exercise 0**.



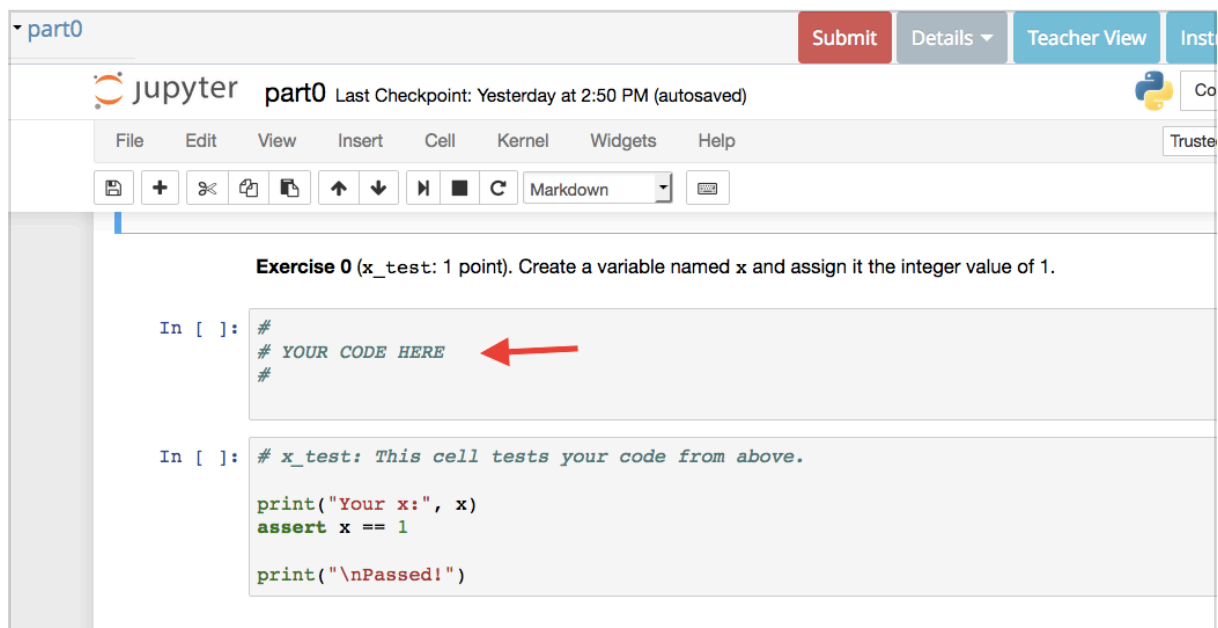
The screenshot shows the JupyterLab interface for 'part0'. The top bar includes a 'Submit' button, 'Details', 'Teacher View', and 'Inst'. The main area displays the exercise title 'Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer value of 1.' Below the title, there are two code cells. The first cell, labeled 'In []:', contains the placeholder text '# YOUR CODE HERE'. A red arrow points to this cell. The second cell, also labeled 'In []:', contains test code: '# x_test: This cell tests your code from above.', 'print("Your x:", x)', 'assert x == 1', and 'print("\nPassed!")'.

```
Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer value of 1.
```

```
In [ ]: #  
# YOUR CODE HERE  
#
```

```
In [ ]: # x_test: This cell tests your code from above.  
print("Your x:", x)  
assert x == 1  
print("\nPassed!")
```

Exercises will typically pose a coding problem for you to solve. Immediately below the problem statement, there will be a code cell where you can type your answer.



This screenshot is identical to the one above, showing the JupyterLab interface for 'part0' with the exercise title and two code cells. A red arrow points to the first code cell, which contains the placeholder text '# YOUR CODE HERE'.

```
Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer value of 1.
```


```
In [ ]: #  
# YOUR CODE HERE  
#
```

```
In [ ]: # x_test: This cell tests your code from above.  
print("Your x:", x)  
assert x == 1  
print("\nPassed!")
```

And immediately below that, there will (usually) be a "test cell," which contains additional code that runs tests to verify your solution.

Exercise 0 (x_test: 1 point). Create a variable named `x` and assign it the i

```
In [ ]: #  
# YOUR CODE HERE  
#
```



```
In [ ]: # x_test: This cell tests your code from above.  
  
print("Your x:", x)  
assert x == 1  
  
print("\nPassed!")
```

Usually, there will be a comment line at the top of the test cell that gives it a "name." In this example, that name is `x_test`. Later, when you submit the job, if any of the test cases fails, the autograder will report this name so you can try to track down what went wrong.

Speaking of errors: Just for fun, try running the test cell **BEFORE** filling in and running the placeholder for your answer.

Exercise 0 (x_test: 1 point). Create a variable named `x` and i

```
In [ ]: #  
# YOUR CODE HERE  
#
```

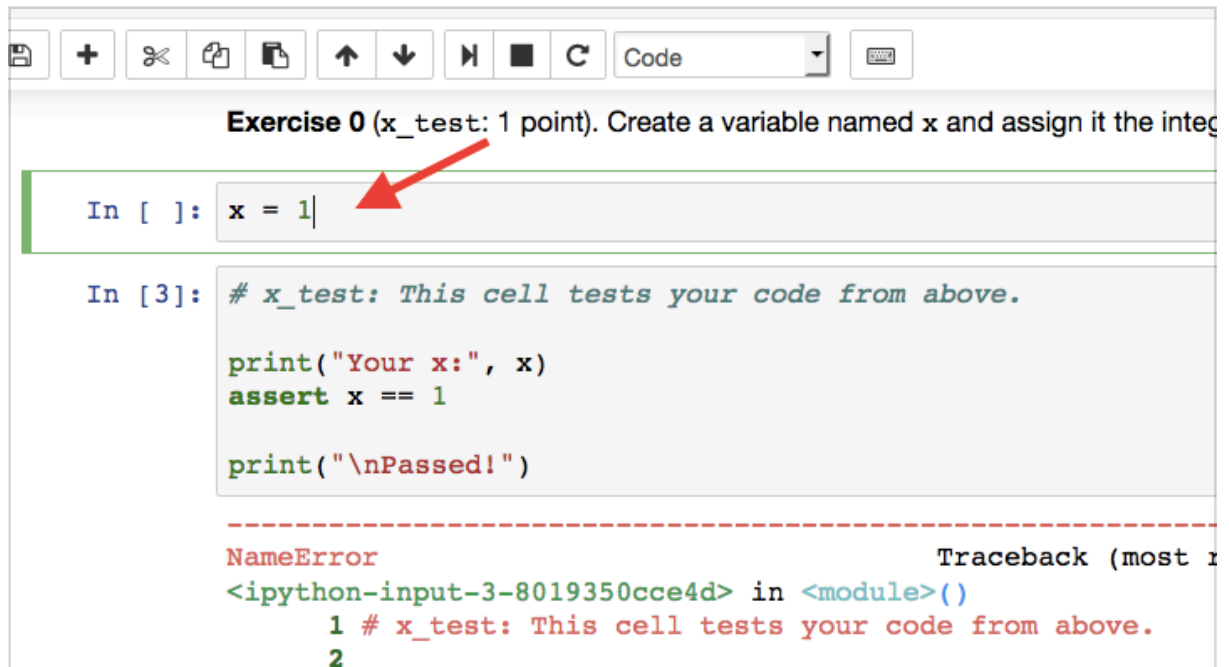
```
In [3]: # x_test: This cell tests your code from above  
  
print("Your x:", x)  
assert x == 1  
  
print("\nPassed!")
```



```
-----  
NameError                                Trac  
<ipython-input-3-8019350cce4d> in <module>()  
1 # x_test: This cell tests your code fr
```

In this case, the test code caught the fact that we hadn't done what we were asked, which was to define a variable named `x` and set its value to 1.

Let's correct the mistake. Replace the Python comments indicating where your code should go with the assignment statement, `x = 1`. That is, start by clicking within the answer cell to get an active cursor within the cell, and then type the solution as indicated below.



The screenshot shows a Jupyter Notebook interface. At the top, there is a toolbar with icons for saving, adding, deleting, copying, pasting, and running code. Below the toolbar, the notebook content is displayed. The first cell is a text prompt: "Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer 1." The second cell is a code input cell with the text "In []: x = 1". A red arrow points to this cell. The third cell is a code input cell with the following text: "In [3]: # x_test: This cell tests your code from above.", "print('Your x:', x)", "assert x == 1", "print('\nPassed!')". Below this cell, a red dashed line separates the code from the error message. The error message is a "NameError" with a traceback showing the error occurred in the second cell. The traceback text is: "NameError Traceback (most recent call last)", "<ipython-input-3-8019350cce4d> in <module>()", "1 # x_test: This cell tests your code from above.", "2".

```
Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer 1.
```

```
In [ ]: x = 1
```

```
In [3]: # x_test: This cell tests your code from above.
```

```
print("Your x:", x)
```

```
assert x == 1
```

```
print("\nPassed!")
```

```
-----
```

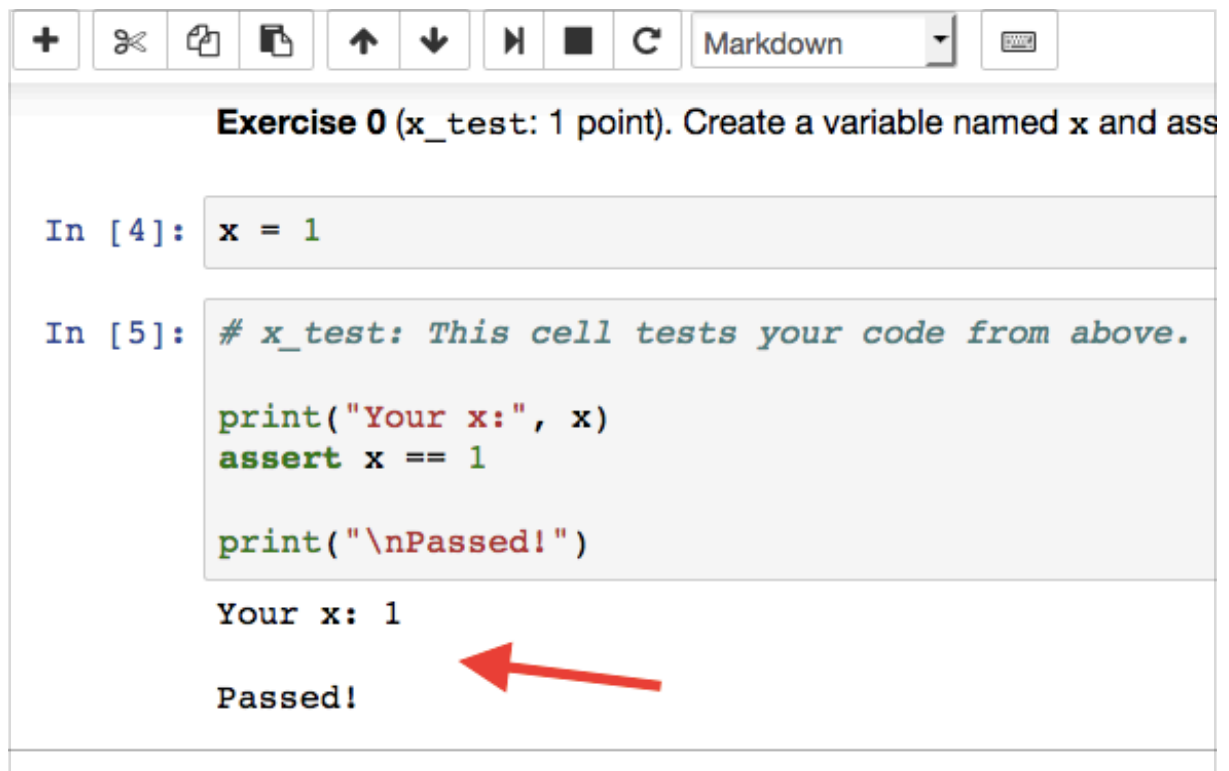
```
NameError Traceback (most recent call last)
```

```
<ipython-input-3-8019350cce4d> in <module>()
```

```
1 # x_test: This cell tests your code from above.
```

```
2
```

Now run that cell and then re-run the test cell.



The screenshot shows the same Jupyter Notebook interface as before, but now the code has been executed successfully. The first cell is the same text prompt: "Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer 1." The second cell is a code input cell with the text "In [4]: x = 1". The third cell is a code input cell with the same text as before: "In [5]: # x_test: This cell tests your code from above.", "print('Your x:', x)", "assert x == 1", "print('\nPassed!')". Below this cell, the output of the code is displayed: "Your x: 1" and "Passed!". A red arrow points to the "Passed!" output.

```
Exercise 0 (x_test: 1 point). Create a variable named x and assign it the integer 1.
```

```
In [4]: x = 1
```

```
In [5]: # x_test: This cell tests your code from above.
```

```
print("Your x:", x)
```

```
assert x == 1
```

```
print("\nPassed!")
```

```
Your x: 1
```

```
Passed!
```

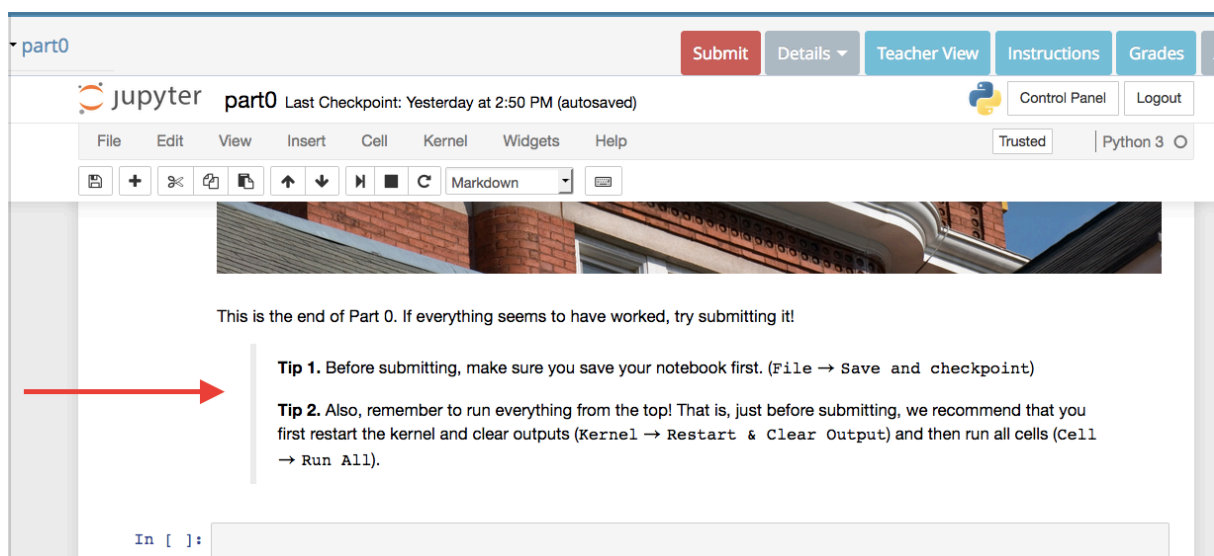
Success!

Observe how the counters change as you execute and re-execute cells.

Go ahead and see if you can complete **Exercise 1** of this part. As indicated in the problem statement, there is no grade assigned to your solution for **Exercise 1**, so if you can't figure it out, don't worry.

Getting ready to submit

Once you are ready, scroll to the bottom of this first notebook for two important tips!



The first tip reminds you to **save your work**. Although Jupyter does periodically autosave the notebook, you can never be too paranoid.

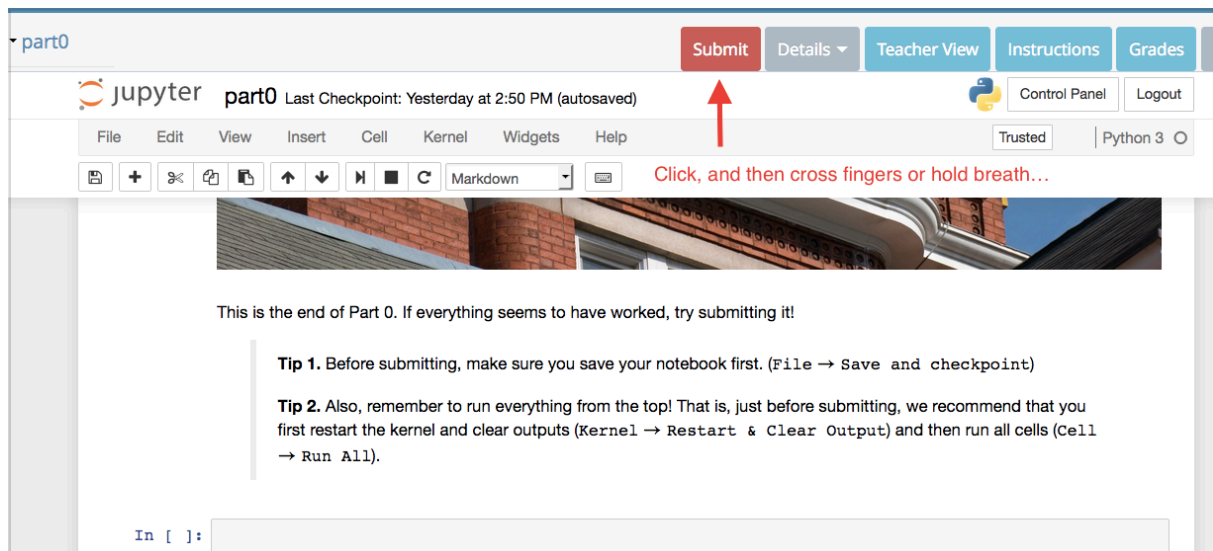
The second tip harkens back to a preceding observation: it's easy in a Jupyter notebook to jump around and, as you are debugging your code, execute and re-execute code cells in many different orders. But when we autograde your notebook, we will run your code from top to bottom. Thus, when you think you have finalized your answer, it's a good idea to clear the execution state and output, and then re-run the entire notebook, as **Tip 2** suggests.

Submitting the notebook

Now for the exciting part: submitting the notebook for automatic grading!

We have configured the system so that you can submit as many times as you wish. This behavior encourages you to test frequently. The score you receive will be the **highest** one you got over all submissions, so you needn't stress out about following a "good" submission by a "bad" one.

When you are ready, click the big red Submit button:



Confirm when prompted. When the system receives your submission, a small box will fade in (and, quickly fade out) of the bottom right corner with the message, Submission recorded.

Submit


Details ▾

Teacher View

Instructions

Grades


Actions ▾



Control Panel

Logout


Trusted | Python 3 ○



ed, try submitting it!

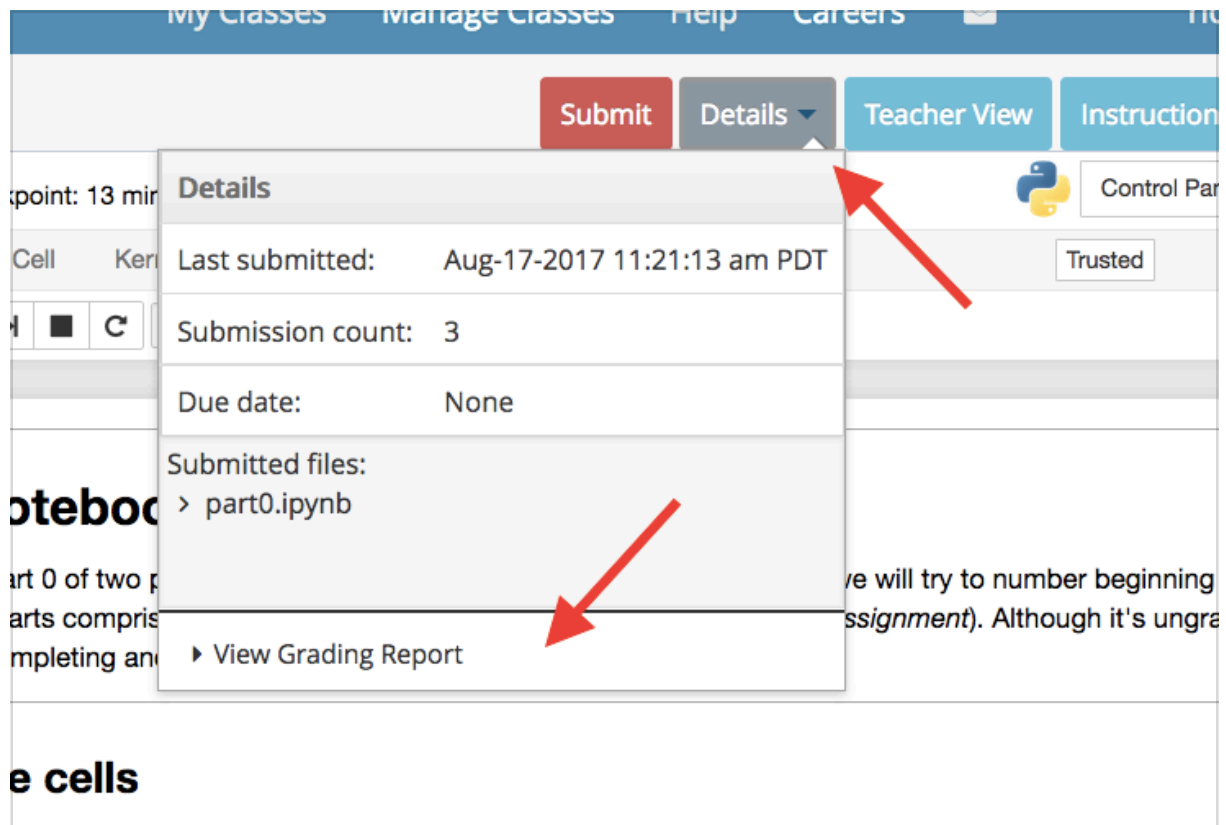
notebook first. (File → Save and checkpoint)

op! That is, just before submitting, we recommend that you
→ Restart & Clear Output) and then run all cells (Cell

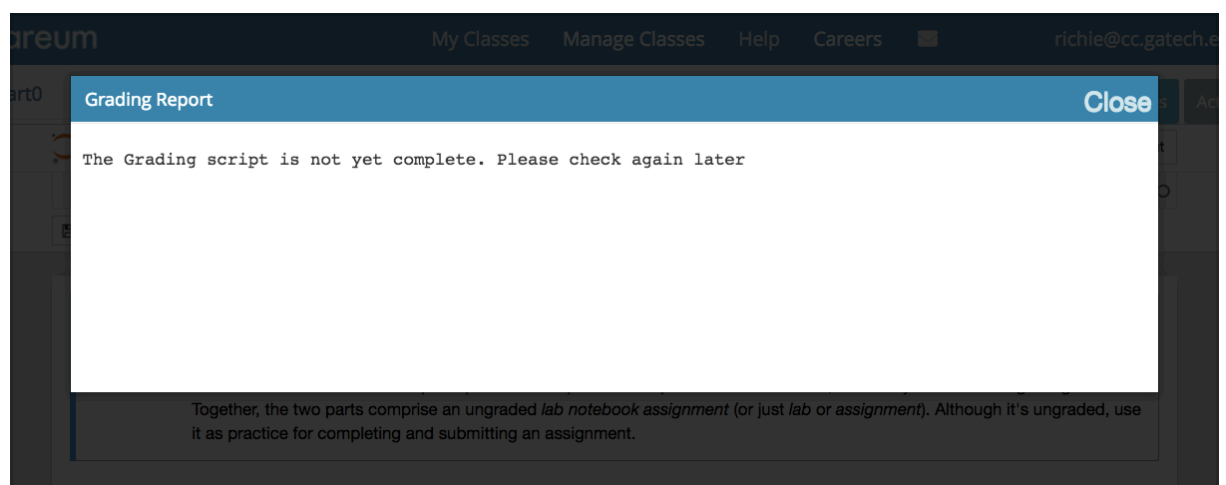


Submission recorded

Your submission is **not** graded right away. To check on its status, click the `Details` menu:



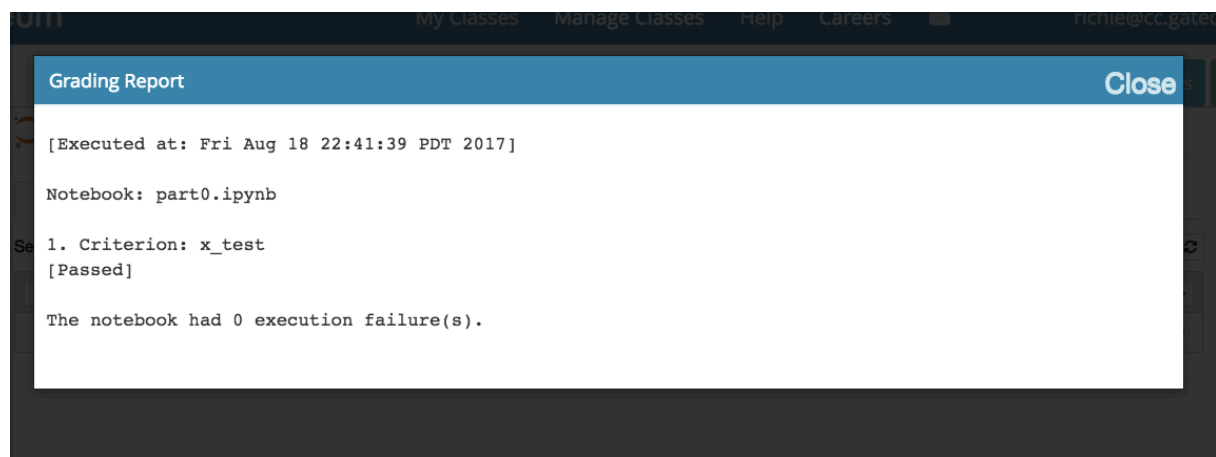
From this dropdown, then click [View Grading Report](#). If you see the message,



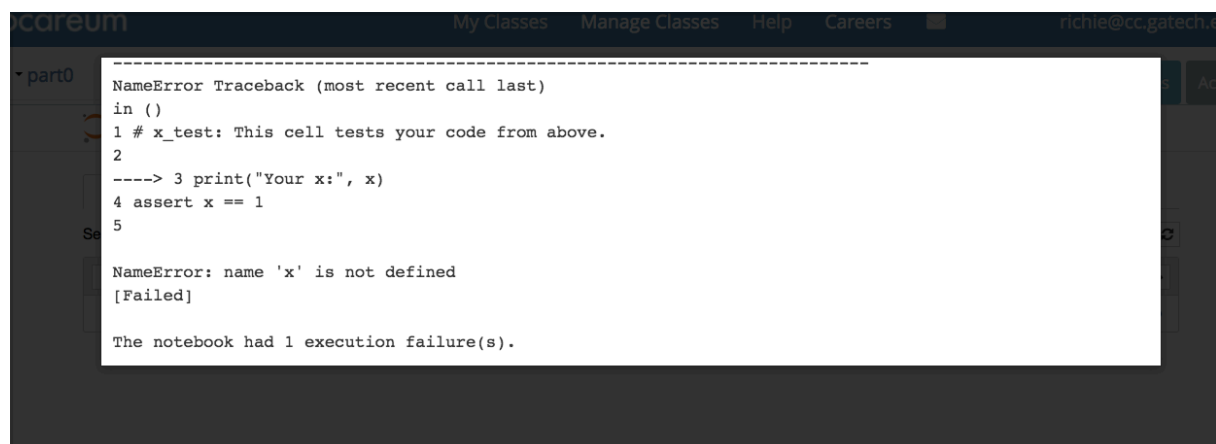
then that means you are still in the queue to be graded. You can close this dialog and repeat [View Grading Report](#) until you see a change in status.

If at first you don't succeed...

Eventually, you will see a report indicating what happened. Here is an example of a passing submission: it indicates that the submitted notebook "had 0 execution failure(s)." Woo hoo!



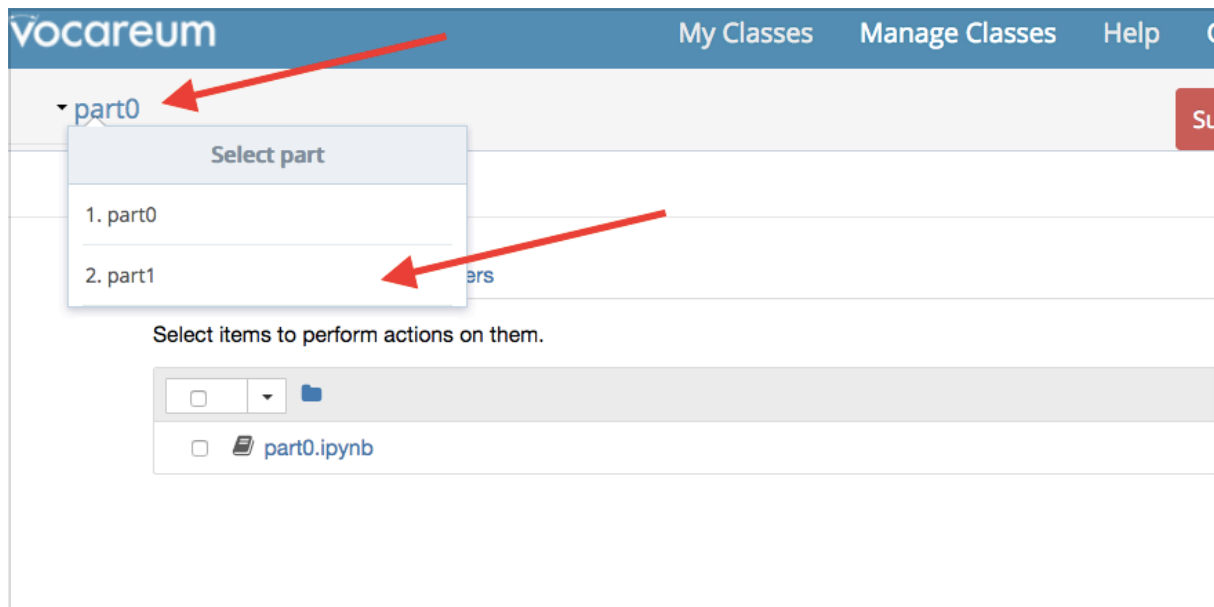
If instead any errors occurred, you'll get a message indicating what test cell(s) failed. For example, here is sample output for a failing submission:



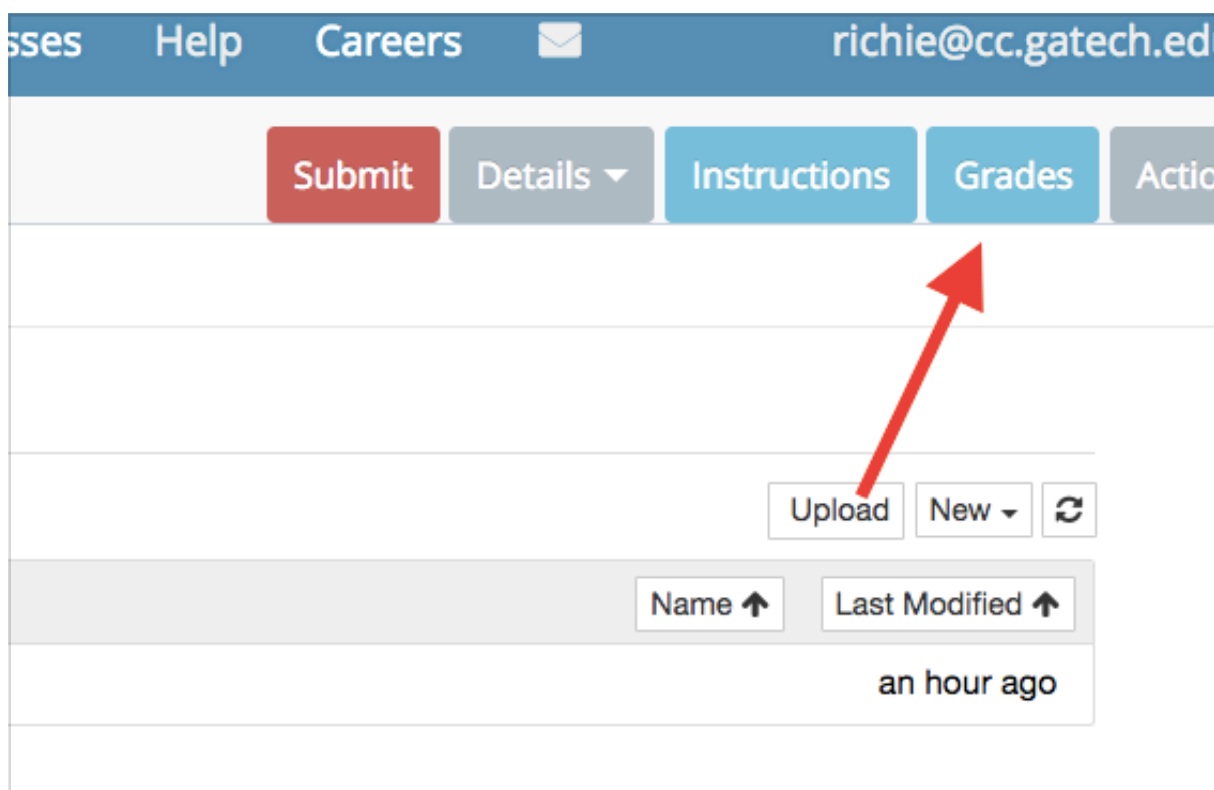
This case mentions the name `x_test`, so you know to go back and look at the corresponding test cell to see if you can figure out where you went wrong.

Wrapping up...

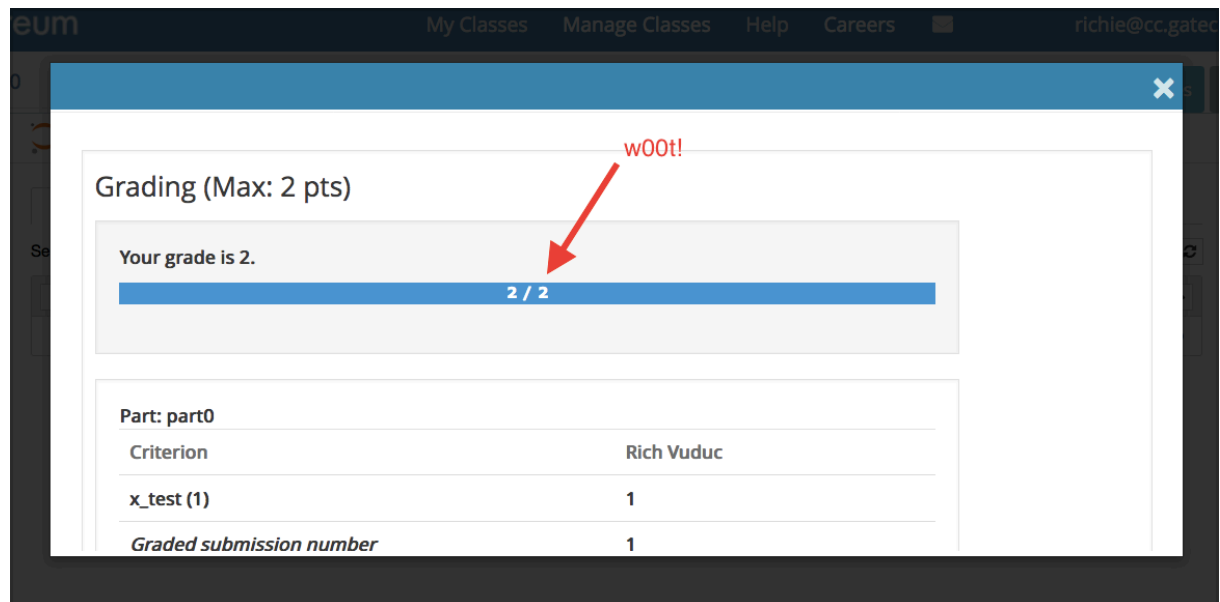
Now that you've done `part0`, see if you can complete `part1`. Recall that you can switch parts through the dropdown menu near the upper-left corner.



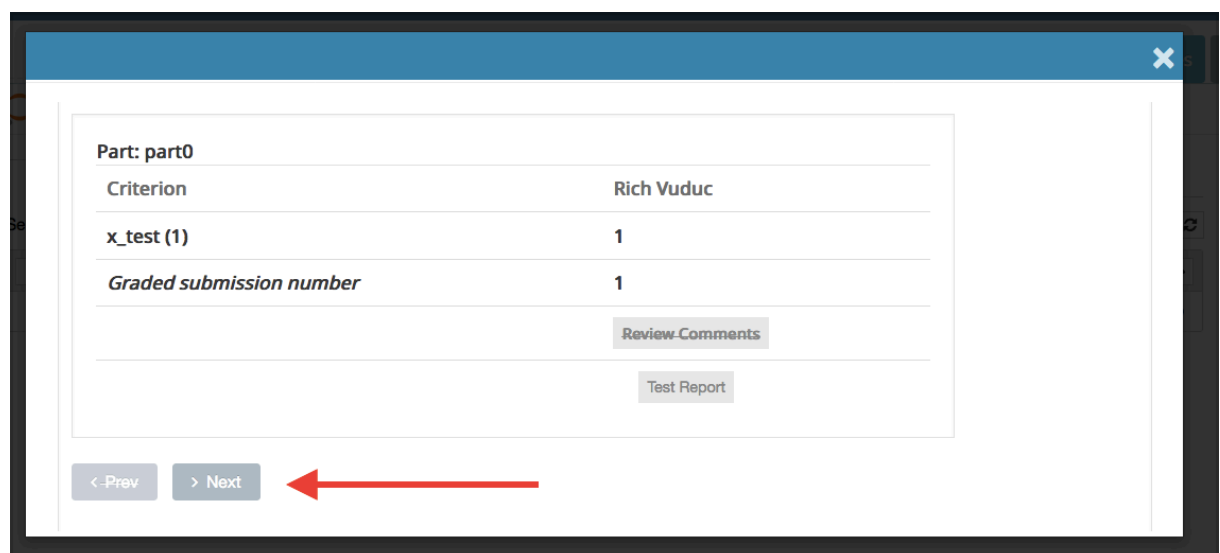
Once you've completed all parts, you might want to verify that you have gotten the expected final score. To see that, click on the **Grades** menu:



For Notebook 0, the total score across all parts is two (2) points, one (1) point for each part. This next screenshot indicates that we got full credit!



This summary screen shows you one part at a time. The bottom of the dialog includes buttons you can use to navigate to other parts. (You may need to scroll to see it.)



Hitting next goes from `part0` to `part1`:

Part: part1

Criterion	Rich Vuduc
filename_test (1)	1
<i>Graded submission number</i>	1

[Review Comments](#)

[Test Report](#)

[< Prev](#) [> Next](#)

You can also click on [Test Report](#) for more details:

Part: part1

Criterion	Rich Vuduc
filename_test (1)	1
<i>Graded submission number</i>	1

[Review Comments](#)

[Test Report](#)

[< Prev](#) [> Next](#)

Test Report

[Close](#)

[Executed at: Thu Aug 17 11:25:39 PDT 2017]

Notebook: part1.ipynb

1. Criterion: filename_test
[Passed]

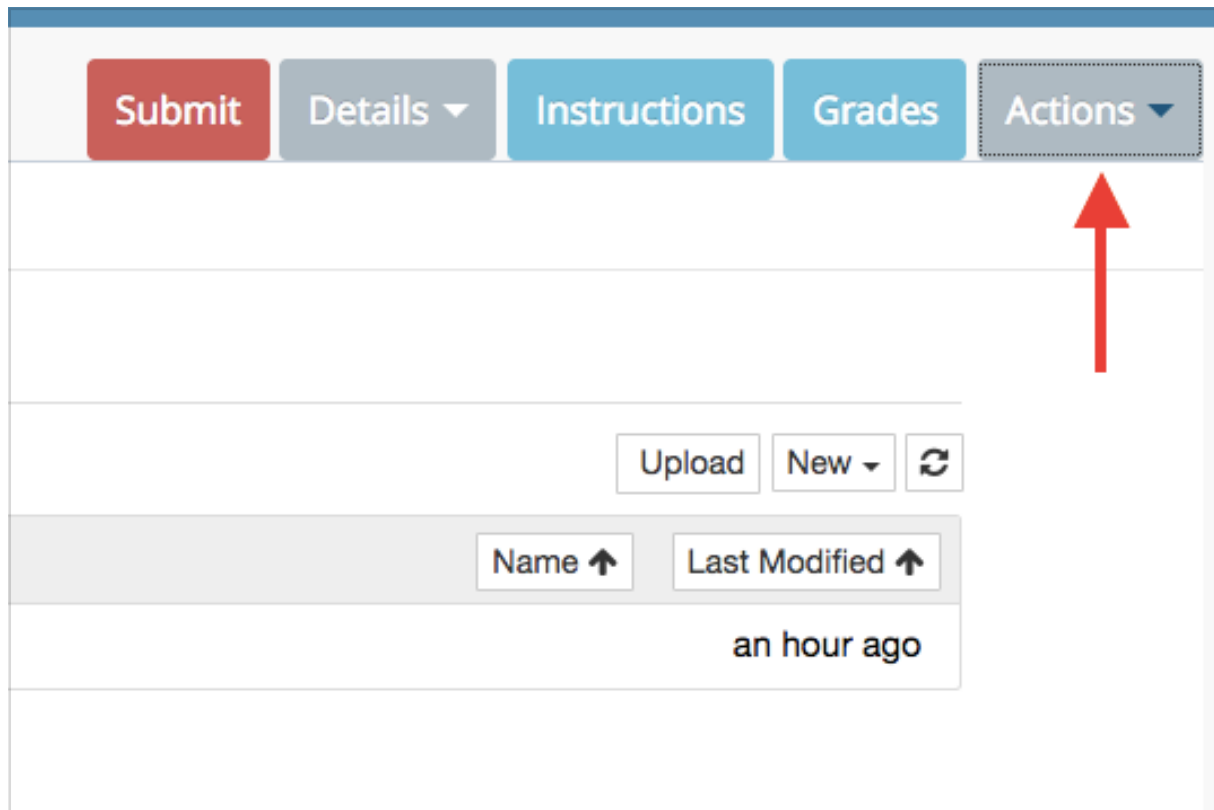
The notebook had 0 execution failure(s).

[< Prev](#)

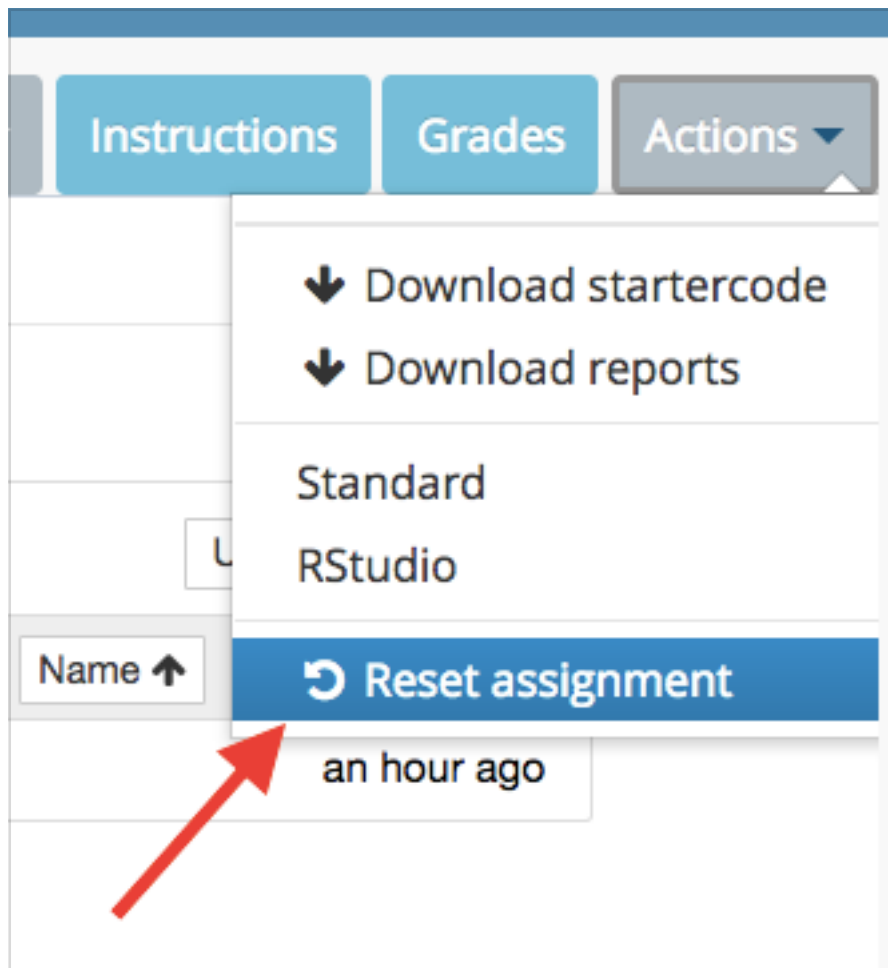
So that's it! If you've reached this point, you may declare victory. The grade book in edX should reflect these results, with a possible small delay.

Ack! I really messed up and just want to start over

You may find yourself in a situation where everything has gone haywire and you just want to start a notebook over, from scratch. Here's how: go to the **Actions** dropdown menu:



Then select **Reset assignment**.



You'll be prompted to confirm.

As always, head to the forums with your questions, and good luck!

[#teaching/cse6040](#)

[#teaching/omsa](#)