I am starting my assignment for Week 3. Remind me how I launch Jupyter Labs on my Mac desktop?

Reasoned for 5 seconds

To launch JupyterLab on your Mac desktop, follow these steps:

1. Open the **Terminal** application (you can find it via Spotlight or in Applications > Utilities).
2. (Optional) If you're using a specific Python environment (e.g., via Anaconda or virtualenv), activate it.
3. Navigate to your project or assignment directory using the `cd` command, for example:

   ```bash
   cd ~/path/to/your/assignment_directory
   ```

4. Type the command below and press **Enter**:

   ```bash
   jupyter lab
   ```

5. JupyterLab will start, and your default web browser should automatically open with the JupyterLab interface. If it doesn't, copy the URL shown in the terminal and paste it into your browser.

This is the standard way to launch JupyterLab on your Mac desktop. Happy coding with your Week 3 assignment!

Why am i unable to search for an extension in the Jupyter Labs PYPI manager? The extension button is there, but the search bas is greyed out. Note that I'm working in my own fork.

Reasoned for 14 seconds

When you see a greyed-out search bar in the extension manager while using your own fork, it's usually because the extension registry isn't properly enabled or configured. In JupyterLab, the extension manager relies on a connection to a registry (or API endpoint) that provides the list of available extensions. In your fork, one or more of the following might be the issue:

- **Registry Configuration:** Your fork might not be pointing to the official registry or a valid alternative. Without a reachable registry, the search functionality won't be available.
- **Disabled Extension Manager:** In some forks or development builds, the extension manager can be intentionally disabled (or only partially enabled) for security or development reasons. Check your configuration settings (such as in your `jupyter_lab_config.json` or build settings) to ensure that the extension manager is fully enabled.
- **Network or API Issues:** If your fork is configured to use a registry URL that isn't responding (or if there are network issues), the search box may appear greyed out as a fallback.

Review your fork's configuration and compare it with the official JupyterLab setup. Make sure that the registry URL is correctly set and that no configuration flags are inadvertently disabling extension searches. This should restore the search functionality in your extension manager.

I enabled the extension manager. Now when I search for it, it says Error searching for extensions: Error: Unhandled error

Reasoned for 7 seconds

This kind of error is often due to a misconfiguration or incomplete setup of the extension registry in your fork. Here are a few things to check:

- **Registry Endpoint:**
  Verify that your fork's configuration points to a valid extension registry. If the registry URL is missing or incorrect in your configuration (often in your JupyterLab config or build settings), the search function won't be able to retrieve any results and can throw an "Unhandled error."
- **Build and Dependency Check:**
  Since you're working with your own fork, ensure that all dependencies are properly installed and that your fork's build is up to date. Sometimes, mismatched versions or missing dependencies can lead to runtime errors.
- **Developer Console:**
  Open your browser's developer console (usually with F12 or right-click → Inspect) when you trigger the search. The detailed error log may provide additional clues about which part of the process is failing.
- **Custom Modifications:**
  If you have made any changes to how the extension manager works in your

fork, double-check that these modifications haven't affected the registry lookup functionality.

Adjusting the registry configuration or reverting any recent changes in your fork's setup should help resolve the issue.

is it because i'm working in JL from here? "http://localhost:8888/lab/tree/Google%20Drive/Education/UM%20Graduate%20School/BGEN632/Week%203/week3labs"

Reasoned for 8 seconds

The folder you're launching JupyterLab from (a Google Drive directory with spaces) is unlikely to be the direct cause of the extension search error. Generally, the extension manager should work regardless of your working directory. However, a few factors to consider are:

- **Path Encoding:**
  While spaces and special characters in a file path can sometimes cause unexpected behavior, the error you're seeing ("Unhandled error") is more often related to misconfigurations or issues with the extension registry itself.
- **Configuration and Registry Setup:**
  Since you're working with your own fork, double-check that the registry URL and other configuration settings are correctly set up. A misconfigured registry endpoint could be the source of the error.
- **Testing in a Different Directory:**
  It might be helpful to launch JupyterLab from a simpler directory (for example, your home directory) to see if the issue persists. If the extension search works there, it might suggest that the path (or related network syncing issues with Google Drive) plays a role, although that's less common.

Overall, while your working directory path might add complexity (e.g., due to spaces), the error is most likely due to configuration issues in your fork or the extension manager itself. Checking the browser's developer console for more detailed logs can also offer further clues.

I may have started my assignment incorrectly. Walk me through the process of forking/syncing/cloning my instructor's workspace in JL for use in GuitHab desktop

Reasoned for 13 seconds

Below is a step-by-step guide to ensure you start your assignment correctly by forking your instructor's workspace, syncing it, and then cloning it using GitHub Desktop:

---

1. Fork the Instructor's Repository

- **Log In & Navigate:**
  Open your web browser, log in to GitHub, and go to your instructor's repository page.
- **Click "Fork":**
  In the upper right corner of the repository page, click the **Fork** button. This creates your own copy of the repository in your GitHub account.

---

2. Clone Your Fork Using GitHub Desktop

- **Open GitHub Desktop:**
  Launch GitHub Desktop and sign in with your GitHub credentials if needed.
- **Clone the Repository:**
  - Click on **File > Clone Repository** (or use the clone button).
  - In the list of repositories, select your fork (it will have the same repository name but now belongs to your account).
  - Choose a local folder where you'd like to store the repository.
  - Click **Clone** to download the repository to your computer.

---

3. Set Up and Sync with the Upstream Repository

To keep your fork up to date with the instructor's repository (often called the "upstream"), you need to add the original repository as an upstream remote.

- **Open Terminal in Your Local Repository:**
  Navigate to your cloned repository folder using the Terminal. For example:

```bash
cd /path/to/your/cloned/repository
```

- **Add the Upstream Remote:**
  Run the following command (replace `instructor-username` and `instructor-repository` with the correct values):

  ```bash
  git remote add upstream https://github.com/instructor-username/instructor-repository.git
  ```

- **Verify the Remotes:**
  List your remotes to ensure the upstream is added:

  ```bash
  git remote -v
  ```

  You should see both `origin` (your fork) and `upstream` (the instructor's repository).

- **Sync Changes from Upstream:**
  When you need to update your fork with the latest changes:

  - **Fetch the latest changes:**

    ```bash
    git fetch upstream
    ```

  - **Merge the changes into your local branch (commonly the main branch):**

    ```bash
    git merge upstream/main
    ```

    (Use `master` instead of `main` if that is the default branch.)

  Alternatively, you can use GitHub Desktop to fetch and merge updates by selecting **Repository > Pull** and choosing the upstream branch if it's configured.

---

4. Start Working on Your Assignment in JupyterLab

- **Launch JupyterLab:**
  Open your Terminal, navigate to your cloned repository directory, and launch JupyterLab:

  ```bash
  cd /path/to/your/cloned/repository jupyter lab
  ```

JupyterLab should open in your default browser with the workspace ready for you to work on your assignment.

- **Commit & Push Your Changes:**
  As you work, make sure to commit your changes using GitHub Desktop (or via command line) and push them to your fork on GitHub. This helps keep your work backed up and easily shareable.

Following these steps should help you correctly start your assignment by forking, syncing, and cloning your instructor's workspace using GitHub Desktop and then launching it in JupyterLab. Happy coding!

How do I delete a previous fork in JL?

Reasoned for 9 seconds

There are two different places where your fork might exist – on GitHub (your online fork) and on your local machine (the cloned repository visible in JupyterLab). Here's how you can delete each:

---

1. Deleting the GitHub Fork

If you want to remove the fork from GitHub entirely, follow these steps:

- **Log In to GitHub:**
  Open your web browser and log in to your GitHub account.
- **Navigate to Your Fork:**
  Go to your forked repository's page (it will be under your account).
- **Access Repository Settings:**
  Click on the **Settings** tab (usually located at the top or in a sidebar).
- **Scroll to the Danger Zone:**
  At the bottom of the Settings page, find the **Danger Zone** section.
- **Delete the Repository:**
  Click on **Delete this repository** and follow the prompts (you'll be asked to type the repository name to confirm).

  > **Warning:** This action is irreversible and will permanently remove your fork from GitHub.

2. Deleting the Local Clone in JupyterLab

If you only need to remove the local copy (the forked repository you cloned to your machine and are viewing in JupyterLab), you can do so using your file system:

- **Using the JupyterLab File Browser:**
  - Locate the folder corresponding to your cloned repository in the JupyterLab file browser.
  - Right-click (or use the appropriate context menu) on the folder and select **Delete**.
  - Confirm the deletion if prompted.
- **Using Finder or Terminal:**
  - Alternatively, close JupyterLab and delete the folder from your local file system using Finder or by running a command like `rm -rf /path/to/your/cloned/repository` in the Terminal.

Choose the method that fits your needs. Deleting the fork on GitHub removes it entirely from your online account, while deleting the local clone only affects your local environment without impacting the remote repository.

> Going back a step, your "Step 3 Set Up and Sync with the Upstream Repository" is different from how I've done this is previous weeks. Isn't there a sync button or something like that?

Reasoned for 12 seconds

GitHub Desktop does include a sync feature, but it mainly handles syncing your fork (origin) with GitHub—not directly syncing with an upstream repository. In previous weeks, if you were only working with your fork and not pulling in updates from your instructor's repository, the built-in sync button may have seemed like the only step you needed.

When you need to update your fork with changes from the instructor's repo (upstream), you must first add the upstream remote manually. Once that's set up, you can use GitHub Desktop's fetch or pull functions to retrieve changes from upstream. Some versions of GitHub Desktop might even offer an "Update from

upstream" option under the Repository menu, but this isn't always as straightforward as the one-click sync you're used to.

So in summary:

- The sync button is for updating your local fork with its origin on GitHub.
- To incorporate upstream changes, you need to add the upstream remote and then fetch and merge those updates—either via GitHub Desktop (if supported) or the command line.

how do i open JL on my desktop again?

Reasoned for 5 seconds

To open JupyterLab on your desktop, follow these steps:

1. **Open Terminal:**
   - On your Mac, open the Terminal application (you can use Spotlight or navigate to Applications > Utilities).
2. **Navigate to Your Project Directory:**
   - Change to the directory where your project or assignment is located. For example:

     bash Copy

     ```
     cd ~/path/to/your/project
     ```
3. **Launch JupyterLab:**
   - Run the command:

     bash Copy

     ```
     jupyter lab
     ```
   - This will start JupyterLab and typically open your default web browser with the JupyterLab interface. If it doesn't open automatically, copy the URL shown in the Terminal and paste it into your browser.

These steps will get JupyterLab running on your desktop again.

it's not working

Reasoned for 6 seconds

Let's try a few troubleshooting steps to diagnose and fix the issue:

1. **Verify JupyterLab Installation:**
   Open your Terminal and run:

   ```bash
   jupyter --version
   ```

   If you don't see version information, JupyterLab might not be installed. You can install it with:

   ```bash
   pip install jupyterlab
   ```

   (Or, if you're using Conda, try `conda install -c conda-forge jupyterlab`.)

2. **Check Your Python Environment:**
   If you're using a virtual environment or Conda, ensure it's activated before running `jupyter lab`. For example:

   ```bash
   conda activate myenv
   ```

   Then try launching JupyterLab again.

3. **Launch Using Python Module:**
   Sometimes invoking it as a module can help:

   ```bash
   python -m jupyter lab
   ```

4. **Look at Terminal Error Messages:**
   Any error output in the Terminal can provide clues. If you see errors related to port conflicts or missing packages, those messages can help pinpoint the issue.

5. **Check for Port Conflicts:**
   By default, JupyterLab runs on port 8888. If something else is using that port, you can specify a different port:

   ```bash
   jupyter lab --port=8889
   ```

6. **Reboot or Try a Different Directory:**
   Occasionally, file path issues or temporary glitches might cause problems. Try launching from a simpler directory (like your home directory) to see if it makes a difference:

   ```bash
   ```

```
cd ~ jupyter lab
```

If you continue to experience issues, please share any error messages you see, and we can further diagnose the problem.

I believe i have successfully forked the repo, but how do i clone it?

Reasoned for 8 seconds

You can clone your forked repository either using GitHub Desktop or via the command line. Here's how:

Cloning with GitHub Desktop

1. **Open GitHub Desktop:**
   Launch GitHub Desktop and log in if necessary.
2. **Access the Clone Option:**
   - Click on **File** in the menu bar, then select **Clone Repository...**
   - Alternatively, use the **Clone a Repository** button if it's visible on your home screen.
3. **Select Your Fork:**
   - In the repository list, find and select your fork (it should appear under your GitHub account).
   - If it doesn't appear automatically, make sure you're viewing repositories under your account.
4. **Choose Local Path:**
   - Pick the folder on your desktop (or any location) where you want to store the repository.
5. **Clone the Repository:**
   Click the **Clone** button. GitHub Desktop will download the forked repo to your chosen location.

Cloning via Command Line

1. **Copy the Repository URL:**
   - Go to your fork's page on GitHub.
   - Click the **Code** button and copy the HTTPS or SSH URL.
2. **Open Terminal:**
   Open the Terminal app on your Mac.
3. **Navigate to Your Desired Directory:**
   Use the `cd` command to change to the folder where you want the repository cloned, for example:

   ```bash
   cd ~/path/to/your/desired/directory
   ```

4. **Clone the Repository:**
   Run the following command (replace `<repository-url>` with your fork's URL):

   ```bash
   git clone <repository-url>
   ```

   This command will create a local copy of your fork in the specified directory.

Once cloned, you can open your project in your favorite editor or launch JupyterLab from the cloned directory. Happy coding!

remind me the diffrence between int and float

Reasoned for 16 seconds

In most programming languages, including Python:

- **int (Integer):**
  Represents whole numbers without a fractional part (e.g., 3, -10, 0).
- **float (Floating Point):**
  Represents numbers that have a decimal point or fractional component (e.g., 3.14, -0.001, 2.0).

Floats are used when you need more precision with decimal values, while ints are used for counting or when you only need whole numbers.

is there a way to export all the code from multiple blocks in a jupyter labs project?

Reasoned for 6 seconds

Yes, you can export all the code cells from a notebook into a Python script. Here are two common methods:

1. **Using the JupyterLab Menu:**
   - Open your notebook in JupyterLab.
   - Go to **File > Export Notebook As...**
   - Select **Export Notebook to Executable Script**.
   - This will create a .py file that contains all the code from your notebook's cells.
2. **Using the Command Line with nbconvert:**
   Open your Terminal, navigate to the directory containing your notebook, and run:

   ```bash
   jupyter nbconvert --to script your_notebook.ipynb
   ```

   This command converts your notebook to a Python script file.

If you have multiple notebooks, you can either export each one individually using these methods or write a small script to batch convert them.

2/2

what does "f"" mean in JL?

Reasoned for a couple of seconds

In Python, the prefix "f" before a string denotes an **f-string** (formatted string literal). This allows you to embed Python expressions directly within the string using curly braces. For example:

```python
name = "Alice" greeting = f"Hello, {name}!"
```

Copy

This will output `"Hello, Alice!"` because the expression `{name}` is replaced with the value of the variable `name`. F-strings are a concise way to format strings and were introduced in Python 3.6.

Here is my first assignment for this week. I've done everything prior to "Comparison of Variables." I would like you to assist me with the rest of the assignment beginning from that instruction, and let's do this one step / one output at a time. "Instructor-led Lab
In this assignment you will create a Python notebook showcasing the concepts you learned about in the tutorial.

Celsius and Fahrenheit Conversion (again)
In this assignment you will revisit your Celsius-to-Fahrenheit converter program. Go ahead and copy the contents of your last Python file into this assignment. For this assignment, you will be using the various operators to compare inputs and calculated values. Additionally, you will introduce the temperature scale Kelvin.

Kelvin Temperature Scale
The conversion from Kelvin to Celsius is more straight forward than converting Celsius to Fahrenheit. The equation is

Celsius = Kelvin - 273.15

In your notebook, perform the following:

Add a variable to assign the value for Kelvin
Write code to convert Celsius to Kelvin; this should occur after your code converts Celsius to Fahrenheit.
Where you output the values for Celsius and Fahrenheit, add Kelvin too. Your output should look like this:

c degrees Celsius is f degrees Fahrenheit which is k degrees Kelvin.

Comparison Operators
Now that your program is able to generate values for Kelvin, you will further extend the program to compare your temperature values to a second set of temperature values.

Second Set of Variables
You will need to create three new variables: celsius_two, fahrenheit_two, and kelvin_two. You will prompt the user for input (i.e., celsius) like you did above and convert it into Fahrenheit and Kelvin. In a code cell, output the new values, similar to what you did previously, only with a slight modification like so:

Second set of values: c degrees Celsius is f degrees Fahrenheit which is k degrees Kelvin.

Comparison of Variables
You will compare the values generated for each of the new temperature variables using the operators in the table below to the original set of temperature values. Thus, you will have three pairs of values to compare.

Operator Name Example
== Equal to x == y
!= Not equal to x != y
> Greater than x > y
< Less than x < y
>= Greater than or equal to x >= y
<= Less than or equal to x <= y
Create code that compares the Celsius values, the Fahrenheit values, and the Kelvin values. The comparison operator is preselected by you, but you should use a variety for all your comparisons. For each comparison, you will output text based on the result of the comparison. You will need to use a combination of if and elif statements.

For example, here is a table that provides some of the logic I might use for my program:

Comparison Result Text
celsius > celsius_two Original value is larger "The original Celsius value of {value} is larger than the new value of {value}."
celsius < celsius_two New value is larger "The original Celsius value of {value} is smaller than the

new value of {value}."
celsius == celsius_two Both are equal "The original Celsius value of {value} equals the new value of {value}."
fahrenheit >= fahrenheit_two Original value is larger "The original Fahrenheit value of {value} is greater than or equal to the new value of {value}."
fahrenheit < fahrenheit_two New value is larger "The original Fahrenheit value of {value} is smaller than the new value of {value}."
fahrenheit == fahrenheit_two Both are equal "The original Fahrenheit value of {value} is equal to {value}."
kelvin <= kelvin_two Original value is larger "The original Kelvin value of {value} is less than or equal to the new value of {value}."
kelvin < kelvin_two New value is larger "The original Kelvin value of {value} is smaller than the new value of {value}."
kelvin != kelvin_two Not equal "The original Kelvin value of {value} is not equal to the new value of {value}."
For example, if I type -40 for my first value of celsius and then type in 0 for my second value, the following will output:

Welcome to my Fahrenheit to Celsius converter!
Please provide a value, in Celsius, to convert:
-40
-40.0 degrees Celsius is -40.0 degrees Fahrenheit which is 233.14999999999998 degrees Kelvin.

Please provide another value, in Celsius, to convert:
0
Second set of values: 0.0 degrees Celsius is 32.0 degrees Fahrenheit which is 273.15 degrees Kelvin.

The original Celsius value of -40.0 is smaller than the new value of 0.0.
The original Fahrenheit value of -40.0 is smaller than the new value of 32.0.
The original Kelvin value of 233.14999999999998 is less than or equal to the new value of 273.15.
If I choose 80 and -10 as my new set of Celsius values, the following is output:

Welcome to my Fahrenheit to Celsius converter!
Please provide a value, in Celsius, to convert:
80
80.0 degrees Celsius is 176.0 degrees Fahrenheit which is 353.15 degrees Kelvin.

Please provide another value, in Celsius, to convert:
-10
Second set of values: -10.0 degrees Celsius is 14.0 degrees Fahrenheit which is 263.15 degrees Kelvin.

The original Celsius value of 80.0 is larger than the new value of -10.0.
The original Fahrenheit value of 176.0 is greater than or equal to the new value of 14.0.
The original Kelvin value of 353.15 is not equal to the new value of 263.15.
Logical Operators
Let's extend your program one more time. We are going to use the logical operators and and or.

You will use one of your Celsius comparisons and one of your Fahrenheit comparisons with the and logical operator to evalute both and output the results. If both are true, output, "Python speaks the truth!" If it is not, ouput, "Python lies." Prior to your comparison, please print to the screen what your comparison is.

Using one of your Celsius comparisons and one of your Kelvin comparisons from above, use the or logical operator. Provide the same output. Prior to your comparison, please print to the screen what your comparison is.

Here is an example of my output:

Welcome to my Fahrenheit to Celsius converter!
Please provide a value, in Celsius, to convert:
-40
-40.0 degrees Celsius is -40.0 degrees Fahrenheit which is 233.14999999999998 degrees Kelvin.

Please provide another value, in Celsius, to convert:
0
Second set of values: 0.0 degrees Celsius is 32.0 degrees Fahrenheit which is 273.15 degrees Kelvin.

The original Celsius value of -40.0 is smaller than the new value of 0.0.
The original Fahrenheit value of -40.0 is smaller than the new value of 32.0.
The original Kelvin value of 233.14999999999998 is less than or equal to the new value of 273.15.

celsius == celsius_two and fahrenheit <

fahrenheit_two
Python lies.

celsius == celsius_two or kelvin != kelvin_two
Python speaks the truth!"

Reasoned for 31 seconds

Let's start with the **Comparison of Variables** step. For this part, we'll assume you've already obtained the following variables from earlier code:

- **First set:**
  `celsius`, `fahrenheit`, `kelvin`
- **Second set:**
  `celsius_two`, `fahrenheit_two`, `kelvin_two`

Now add a new code cell with the following code to compare these values using a variety of comparison operators:

python Copy

```
# Comparison of Celsius values using >, <, and == if celsius > celsius_two: print(f"The original Celsius value of {celsius} is larger than the new value of {celsius_two}.") elif celsius < celsius_two: print(f"The original Celsius value of {celsius} is smaller than the new value of {celsius_two}.") else: print(f"The original Celsius value of {celsius} equals the new value of {celsius_two}.") # Comparison of Fahrenheit values using >= and < if fahrenheit >= fahrenheit_two: print(f"The original Fahrenheit value of {fahrenheit} is greater than or equal to the new value of {fahrenheit_two}.") else: print(f"The original Fahrenheit value of {fahrenheit} is smaller than the new value of {fahrenheit_two}.") # Comparison of Kelvin values using <= (as in the examples) if kelvin <= kelvin_two: print(f"The original Kelvin value of {kelvin} is less than or equal to the new value of {kelvin_two}.") else: print(f"The original Kelvin value of {kelvin} is greater than the new value of {kelvin_two}.")
```

This cell uses:

- For Celsius: `>`, `<`, and `==`
- For Fahrenheit: `>=` and `<`
- For Kelvin: the `<=` operator

When you run this cell, you'll see output text comparing the original and new values for each temperature scale.

Let me know once you've reviewed or run this step, and then we can move on to incorporating the **Logical Operators** portion!

Remove the commentary from the code and revise; do not include commentary in any of our code.

Reasoned for 6 seconds

python Copy

```
if celsius > celsius_two: print(f"The original Celsius value of {celsius} is larger than the new value of {celsius_two}.") elif celsius < celsius_two: print(f"The original Celsius value of {celsius} is smaller than the new value of {celsius_two}.") else: print(f"The original Celsius value of {celsius} equals the new value of {celsius_two}.") if fahrenheit >= fahrenheit_two: print(f"The original Fahrenheit value of {fahrenheit} is greater than or equal to the new value of {fahrenheit_two}.") else: print(f"The original Fahrenheit value of {fahrenheit} is smaller than the new value of {fahrenheit_two}.") if kelvin <= kelvin_two: print(f"The original Kelvin value of {kelvin} is less than or equal to the new value of {kelvin_two}.") else: print(f"The original Kelvin value of {kelvin} is greater than the new value of {kelvin_two}.")
```

now let's do the Logical Operators portion of the assignment

Reasoned for 5 seconds

python Copy

```
print("celsius == celsius_two and fahrenheit < fahrenheit_two") if celsius == celsius_two and fahrenheit < fahrenheit_two: print("Python speaks the truth!") else: print("Python lies.") print("celsius == celsius_two or kelvin != kelvin_two") if celsius == celsius_two or kelvin != kelvin_two: print("Python speaks the truth!") else: print("Python lies.")
```

what does "==" mean in python?

Reasoned for 7 seconds

In Python, "==" is the equality operator. It checks if the values on both sides are equal and returns a Boolean result—either True if they are equal or False if they are not. For example, `5 == 5` evaluates to True, whereas `5 == 3` evaluates to False.

so is the first comparison between celsius 1 or 2, and fahrenheit 1 or 2?

Reasoned for a few seconds

Yes, exactly. In that comparison, the variable `celsius` (from the first set) is compared to `celsius_two` (from the second set), and `fahrenheit` (from the first set) is compared to `fahrenheit_two` (from the second set).

But the instructions say "You will use one of your Celsius comparisons and one of your Fahrenheit comparisons"

Reasoned for 5 seconds

The instructions mean that when you create your logical operator comparisons, you should combine one condition that compares the Celsius values with another condition that compares either the Fahrenheit values (for the "and" case) or the Kelvin values (for the "or" case). For example, in the code:

```python
if celsius == celsius_two and fahrenheit < fahrenheit_two:
print("Python speaks the truth!") else: print("Python lies.")
```

- The first part (`celsius == celsius_two`) is one of your Celsius comparisons.
- The second part (`fahrenheit < fahrenheit_two`) is one of your Fahrenheit comparisons.

You are not mixing the variables (i.e., comparing a Celsius value directly with a Fahrenheit value), but rather evaluating two separate conditions—one for Celsius and one for Fahrenheit—and then combining them with the logical operator `and` .

Similarly, for the `or` example:

```python
if celsius == celsius_two or kelvin != kelvin_two: print("Python
speaks the truth!") else: print("Python lies.")
```

- The first condition is a Celsius comparison.
- The second is a Kelvin comparison.

This is what the instructions mean by using one comparison from the Celsius set and one from the Fahrenheit (or Kelvin) set in your logical operations.

I'm unclear on the output of our instructor's model vs my own assignment. My responses to the various prompts are piecemeal, e.g. one output after each prompt. but my instructor suggests the total output should be unified in a single output, like this:
"Welcome to my Fahrenheit to Celsius converter!
Please provide a value, in Celsius, to convert:
-40
-40.0 degrees Celsius is -40.0 degrees Fahrenheit which is 233.14999999999998 degrees Kelvin.

Please provide another value, in Celsius, to convert:
0
Second set of values: 0.0 degrees Celsius is 32.0 degrees Fahrenheit which is 273.15 degrees Kelvin.

The original Celsius value of -40.0 is smaller than the new value of 0.0.
The original Fahrenheit value of -40.0 is smaller than the new value of 32.0.
The original Kelvin value of 233.14999999999998 is less than or equal to the new value of 273.15.

celsius == celsius_two and fahrenheit < fahrenheit_two
Python lies.

celsius == celsius_two or kelvin != kelvin_two
Python speaks the truth!"

Reasoned for a few seconds

The instructor's sample output is a unified, continuous sequence of messages that would appear when the entire notebook is run. In your case, if you execute each code cell separately, you might see outputs in stages. However, when you run the whole notebook from top to bottom, all of the outputs are displayed together in the order of execution, creating that unified look.

For example, if you print the welcome message, then prompt and show conversion outputs, and finally print your comparison results in subsequent cells, the notebook will display all these outputs one after the other, just like the instructor's example.

If you prefer to see everything together in one output, you could combine all your code into a single cell. Ultimately, both approaches are acceptable as long as the final combined output matches the instructor's expected result.

Is there code we can put at the end that aggregates all the prior outputs into a single block?

Reasoned for 18 seconds

You can aggregate outputs by capturing your messages in a variable (for example, in a list or a single string) rather than printing them immediately. Then, at the end of your notebook, you print the combined result. Here's an example using a list:

python Copy

```
output_lines = [] def add_line(line): output_lines.append(line)
add_line("Welcome to my Fahrenheit to Celsius converter!")
add_line("Please provide a value, in Celsius, to convert:") celsius
= float(input()) # simulate input if running interactively # ...
perform conversions ... fahrenheit = celsius * 1.8 + 32 kelvin =
celsius + 273.15 add_line(f"{celsius} degrees Celsius is
{fahrenheit} degrees Fahrenheit which is {kelvin} degrees Kelvin.")
add_line("Please provide another value, in Celsius, to convert:")
celsius_two = float(input()) fahrenheit_two = celsius_two * 1.8 + 32
kelvin_two = celsius_two + 273.15 add_line(f"Second set of values:
{celsius_two} degrees Celsius is {fahrenheit_two} degrees Fahrenheit
which is {kelvin_two} degrees Kelvin.") # Example comparisons if
celsius > celsius_two: add_line(f"The original Celsius value of
{celsius} is larger than the new value of {celsius_two}.") elif
celsius < celsius_two: add_line(f"The original Celsius value of
{celsius} is smaller than the new value of {celsius_two}.") else:
add_line(f"The original Celsius value of {celsius} equals the new
value of {celsius_two}.") if fahrenheit >= fahrenheit_two:
add_line(f"The original Fahrenheit value of {fahrenheit} is greater
than or equal to the new value of {fahrenheit_two}.") else:
add_line(f"The original Fahrenheit value of {fahrenheit} is smaller
than the new value of {fahrenheit_two}.") if kelvin <= kelvin_two:
add_line(f"The original Kelvin value of {kelvin} is less than or
equal to the new value of {kelvin_two}.") else: add_line(f"The
original Kelvin value of {kelvin} is greater than the new value of
{kelvin_two}.") add_line("") add_line("celsius == celsius_two and
fahrenheit < fahrenheit_two") if celsius == celsius_two and
fahrenheit < fahrenheit_two: add_line("Python speaks the truth!")
else: add_line("Python lies.") add_line("") add_line("celsius ==
celsius_two or kelvin != kelvin_two") if celsius == celsius_two or
kelvin != kelvin_two: add_line("Python speaks the truth!") else:
add_line("Python lies.") print("\n".join(output_lines))
```

In this example, each message is added to the list `output_lines` using the helper function `add_line()`. At the end, all the messages are printed together as a single block. This way, you can simulate the unified output that your instructor expects.

let's revise our code for this part, as I've manually revised much of the code before it: "Logical Operators
Let's extend your program one more time. We are going to use the logical operators and and or.

You will use one of your Celsius comparisons and one of your Fahrenheit comparisons with the and logical operator to evalute both and output the results. If both are true, output, "Python speaks the truth!" If it is not, ouput, "Python lies." Prior to your comparison, please print to the screen what your comparison is.

Using one of your Celsius comparisons and one of your Kelvin comparisons from above, use the or logical operator. Provide the same output. Prior to your comparison, please print to the screen what your comparison is.

Here is an example of my output:"

Reasoned for 10 seconds

python Copy

```
logical_outputs = [] logical_outputs.append("celsius == celsius_two and fahrenheit < fahrenheit_two") if celsius == celsius_two and fahrenheit < fahrenheit_two: logical_outputs.append("Python speaks the truth!") else: logical_outputs.append("Python lies.") logical_outputs.append("celsius == celsius_two or kelvin != kelvin_two") if celsius == celsius_two or kelvin != kelvin_two: logical_outputs.append("Python speaks the truth!") else: logical_outputs.append("Python lies.") print("\n".join(logical_outputs))
```

Early in my code, I have: "celsius = text" I just got the error "---------- ---------- NameError Traceback (most recent call last) Cell In[2], line 1 ----> 1 celsius = text

NameError: name 'text' is not defined"

Reasoned for a couple of seconds

The error means that Python doesn't recognize the name `text` because it hasn't been defined anywhere in your code. If you intended to assign a string value to `celsius`, you need to enclose the text in quotes. For example:

```python
celsius = "text"
```

If you meant to capture user input, you should use the `input()` function like this:

```python
celsius = input("Please enter the Celsius value: ")
```

And if you need it as a number, you might convert it to a float:

```python
celsius = float(input("Please enter the Celsius value: "))
```

This should resolve the error you're seeing.

I have this before it: "##### text = float(input("Please provide a value, in Celsius, to convert:\n"))"

Reasoned for a few seconds

It looks like the line that assigns the value to the variable is commented out. The "#####" at the beginning makes the line a comment, so it isn't executed and the variable `text` never gets defined. To fix this, remove the "#####" so the code runs. For example:

```python
text = float(input("Please provide a value, in Celsius, to convert:\n")) celsius = text
```

This way, `text` is defined from the user input and then assigned to `celsius` without triggering a NameError.

"text = float(input("Please provide a value, in Celsius, to convert:\n"))" "celsius = text" "----------

---

NameError Traceback (most recent call last)
Cell In[2], line 1
----> 1 celsius = text

NameError: name 'text' is not defined"

Reasoned for a few seconds

This error indicates that Python doesn't see a variable named `text` when you try to assign its value to `celsius`. Here are a few things to check:

- Ensure that the cell containing

  ```python
  text = float(input("Please provide a value, in Celsius, to convert:\n"))
  ```

  was executed before you run the cell with `celsius = text`.
- Make sure the code defining `text` isn't commented out. Even a stray character (like `#` or `#####`) will cause Python to ignore that line.
- Confirm you're running your code in the same session (or kernel). If the kernel restarted or if the cells are in a different notebook, the variable won't persist.

Double-check these points, run the input cell, then run the cell assigning `celsius = text`, and it should work as expected.

what does ":\n" mean?

Reasoned for a couple of seconds

The `":\n"` in the string is composed of two parts:

- The colon (`:`) is simply a character in the prompt.
- The `\n` is a newline character, which causes the cursor to move to the next line after displaying the prompt.

So, when you run the code:

```python
text = float(input("Please provide a value, in Celsius, to convert:\n"))
```

The prompt appears as:

```css
```

```
Please provide a value, in Celsius, to convert:
```

with the cursor on the next line waiting for your input.

Next assignment for this week: "# Independent Lab

In this assignment you will create a Python notebook showcasing the concepts you learned about in the tutorial.

Self-Service Kiosk Logic

In this assignment you will build logic for a self-service kiosk. While the logic for an actual kiosk machine is a little more complex than what you will create, in essence, your program will capture the foundational aspects. This includes product selection, quantity, pricing, and payment. You can set up your kiosk to enable purchase of any type of product. The examples provided below are for a kiosk that sells day tickets and season passes for a ski resort.

Assume the following aspects about your kiosk:

* It sells 5 different products, your choice.
  * Each product has its own price.
  * While two products can have the same listed price, the variables should be different. Display the prices to the user along with the product names using string interpolation.
  * The names of the products and their prices should come from variables. Thus, you will have a total of 5 variables for product names, 5 variables for prices.
* Your program will prompt the user to select a product. It will then prompt them to enter the quantity desired.
  * Confirm what the customer selected and the quantity by outputting the product and quantity to the screen using string interpolation.
* The user can select multiple products with multiple quantities.
  * Additionally, the user can repeat a product order (i.e., the user places an order for 3 pizzas and then decides to add 2 more).
* The user is given the total cost of their order after each selection.

* The program should prompt the user when to complete (i.e., end) their order.
  * The program will confirm which items the customer has ordered along with the quantity.
  * The user *is also given* the total at the completion of their order.
* Your `if-else` statement must have a default or fall-back option that captures any incorrect input from the user. This statement will output, "That is not an option. Please try again."

Your program will need to rely on an infinite loop to continuously provide the user access to the menu selection prompt. This means that when the customer completes their order, you will need to implement a jump statement that ends the infinite loop.

Also, keep in mind that when your program reads in user input, it is stored as a string. So, even if a user enters a number, that number is stored as a string, not an integer or float.

Below is an example from the key of what your program may look like. I created a table to display the products to the customer; *you do not need to recreate it*. I just wanted to display the information in a nicely formatted package.

Start of kiosk order:

```
Welcome! This kiosk sells tickets and pa
Please place your order by selecting the

Here are the ticket options:

|   | Product                       | P
| 1 | Single Day Ticket - Weekday   | $
| 2 | Single Day Ticket - Weekend   | $
| 3 | Season Pass - Adult           | $
| 4 | Season Pass - Child           | $
| 5 | Season Pass - Senior          | $


What would you like to purchase?
1

How many would you like to purchase?
3

You want 3 of Single Day Ticket - Weekda
```

```
The current total for your order is $150
```

Continuation of kiosk order. Customer selects additional prodcuts:

```
You want 3 of Single Day Ticket - Weekda

The current total for your order is $150

If you are finished ordering, please inp
2

Adding to your order...


What would you like to order?
4

How many would you like to order?
1
```

```
You want 1 of Season Pass - Child.

The current total for your order is $609

If you are finished ordering, please inp
1
```

The order is complete and a summary is displayed:

```
The current total for your order is $609

If you are finished ordering, please inp
1

Here is a summary of your final order:

| Product                       | Quant
| Single Day Ticket - Weekday   |     3
| Single Day Ticket - Weekend   |     0
| Season Pass - Adult           |     0
| Season Pass - Child           |     1
| Season Pass - Senior          |     0

The total cost for your order is $609.00
```

" Again, let's take this one step at a time so I can learn more about coding in python/JL

Reasoned for 10 seconds

python Copy

```
product1_name = "Single Day Ticket - Weekday" product1_price =
50.00 product2_name = "Single Day Ticket - Weekend" product2_price =
78.00 product3_name = "Season Pass - Adult" product3_price = 469.00
product4_name = "Season Pass - Child" product4_price = 259.00
product5_name = "Season Pass - Senior" product5_price = 459.00
print("Welcome! This kiosk sells tickets and passes for your local
ski resort.") print("Please place your order by selecting the
product and subsequently providing the quantity desired.\n")
print("Here are the ticket options:\n") print(f"| 1 |
{product1_name:30} | ${product1_price:6.2f} |") print(f"| 2 |
{product2_name:30} | ${product2_price:6.2f} |") print(f"| 3 |
{product3_name:30} | ${product3_price:6.2f} |") print(f"| 4 |
{product4_name:30} | ${product4_price:6.2f} |") print(f"| 5 |
{product5_name:30} | ${product5_price:6.2f} |")
```

Let me know when you're ready to move on to the next step.

Hold on! Let's not use the ski resort tickets, let's do something fun and absurdist; something like George Saunders would write. But nothing offensive.

Reasoned for 4 seconds

python Copy

```
product1_name = "Quantum Lemonade" product1_price = 3.50
product2_name = "Philosopher's Chair" product2_price = 199.99
product3_name = "Cosmic Bubble Tea" product3_price = 4.25
product4_name = "Eternal Notebook" product4_price = 9.99
product5_name = "Time-Traveling Umbrella" product5_price = 49.95
print("Welcome! This kiosk sells bizarre treasures and whimsical
wonders.") print("Please place your order by selecting the product
and subsequently providing the quantity desired.\n") print("Here are
the enchanting options:\n") print(f"| 1 | {product1_name:30} |
${product1_price:6.2f} |") print(f"| 2 | {product2_name:30} |
${product2_price:6.2f} |") print(f"| 3 | {product3_name:30} |
${product3_price:6.2f} |") print(f"| 4 | {product4_name:30} |
${product4_price:6.2f} |") print(f"| 5 | {product5_name:30} |
${product5_price:6.2f} |")
```

that's not very George Saunders-esque. It should evoke feelings of socio-economic crisis and total

lack of upward mobility, but in a lighthearted, absurdist sort of way

Reasoned for 21 seconds

python Copy

```
product1_name = "Bare-Bones Brew" product1_price = 2.00
product2_name = "Downshift Doughnut" product2_price = 1.50
product3_name = "Stagnation Snack Pack" product3_price = 3.25
product4_name = "Crumpled Aspirations Candy" product4_price = 0.75
product5_name = "Subsidized Serenity Smoothie" product5_price = 4.00
print("Welcome! This kiosk offers a taste of humble reality in a
world where upward mobility is but a distant memory.") print("Place
your order by selecting the product and specifying the quantity you
dare to indulge in.\n") print("Behold the offerings:\n") print(f"| 1
| {product1_name:30} | ${product1_price:6.2f} |") print(f"| 2 |
{product2_name:30} | ${product2_price:6.2f} |") print(f"| 3 |
{product3_name:30} | ${product3_price:6.2f} |") print(f"| 4 |
{product4_name:30} | ${product4_price:6.2f} |") print(f"| 5 |
{product5_name:30} | ${product5_price:6.2f} |")
```

This isn't good. Let's just do movie tickets and concessions

Reasoned for 17 seconds

python Copy

```
product1_name = "Standard Movie Ticket" product1_price = 12.00
product2_name = "3D Movie Ticket" product2_price = 15.00
product3_name = "IMAX Ticket" product3_price = 20.00 product4_name =
"Popcorn Combo" product4_price = 8.50 product5_name = "Soft Drink"
product5_price = 3.00 print("Welcome! This kiosk sells movie tickets
and concessions for your cinematic experience.") print("Please place
your order by selecting the product and specifying the quantity
desired.\n") print("Here are the options:\n") print(f"| 1 |
{product1_name:25} | ${product1_price:6.2f} |") print(f"| 2 |
{product2_name:25} | ${product2_price:6.2f} |") print(f"| 3 |
{product3_name:25} | ${product3_price:6.2f} |") print(f"| 4 |
{product4_name:25} | ${product4_price:6.2f} |") print(f"| 5 |
{product5_name:25} | ${product5_price:6.2f} |")
```

ok, next step in the assignment

Reasoned for 14 seconds

python

Copy

```
choice = int(input("What would you like to purchase? (Enter the product number):\n")) qty = int(input("How many would you like to purchase?\n")) if choice == 1: selected_product = product1_name product_price = product1_price elif choice == 2: selected_product = product2_name product_price = product2_price elif choice == 3: selected_product = product3_name product_price = product3_price elif choice == 4: selected_product = product4_name product_price = product4_price elif choice == 5: selected_product = product5_name product_price = product5_price else: print("That is not an option. Please try again.") selected_product = None if selected_product is not None: print(f"You want {qty} of {selected_product}.") current_total = qty * product_price print(f"The current total for your order is ${current_total:6.2f}")
```

revise that one so that: "* The user can select multiple products with multiple quantities. * Additionally, the user can repeat a product order (i.e., the user places an order for 3 pizzas and then decides to add 2 more).
* The user is given the total cost of their order after each selection.
* The program should prompt the user when to complete (i.e., end) their order.
* The program will confirm which items the customer has ordered along with the quantity.
* The user *is also given* the total at the completion of their order.
* Your `if-else` statement must have a default or fall-back option that captures any incorrect input from the user. This statement will output, "That is not an option. Please try again."

Reasoned for 14 seconds

python

Copy

```
order_quantities = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0} order_total = 0.0 while True: choice = input("What would you like to purchase? (Enter the product number, or 'done' to finish):\n") if choice.lower() == "done": break try: choice = int(choice) except: print("That is not an option. Please try again.") continue if choice not in [1, 2, 3, 4, 5]: print("That is not an option. Please try again.") continue try: qty = int(input("How many would you like to purchase?\n")) except: print("That is not an option. Please try again.") continue if choice == 1: product_name = product1_name product_price = product1_price elif choice == 2: product_name = product2_name product_price = product2_price elif choice == 3: product_name = product3_name product_price = product3_price elif choice == 4: product_name = product4_name product_price = product4_price elif choice == 5: product_name = product5_name product_price =
```

```
product5_price order_quantities[choice] += qty order_total += qty *
product_price print(f"You want {order_quantities[choice]} of
{product_name}.") print(f"The current total for your order is
${order_total:6.2f}\n") print("\nHere is a summary of your final
order:\n") print("| Product | Quantity |") print(f"|
{product1_name:30} | {order_quantities[1]:8} |") print(f"|
{product2_name:30} | {order_quantities[2]:8} |") print(f"|
{product3_name:30} | {order_quantities[3]:8} |") print(f"|
{product4_name:30} | {order_quantities[4]:8} |") print(f"|
{product5_name:30} | {order_quantities[5]:8} |") print(f"\nThe total
cost for your order is ${order_total:6.2f}")
```

Is this part from the instructions effectively addressed? "Your program will need to rely on an infinite loop to continuously provide the user access to the menu selection prompt. This means that when the customer completes their order, you will need to implement a jump statement that ends the infinite loop."

Reasoned for 4 seconds

Yes, it is. The code uses a `while True:` loop to continuously prompt the user, and when the user types "done" the `break` statement exits the loop. This effectively meets the requirement of using an infinite loop along with a jump statement to end it.