

## CS 61A: Step-by-Step Debugging Walkthrough

Oftentimes, you'll encounter errors with your code, causing your program to behave unexpectedly or crash altogether. It can be difficult to find and eliminate these errors, but it's important to learn how to do this effectively.

Lucky for us, we have the Python traceback: a report telling us exactly what went wrong in our code and why! Although the traceback output may look slightly intimidating at first, it's much simpler when broken down. This will help us debug step-by-step.

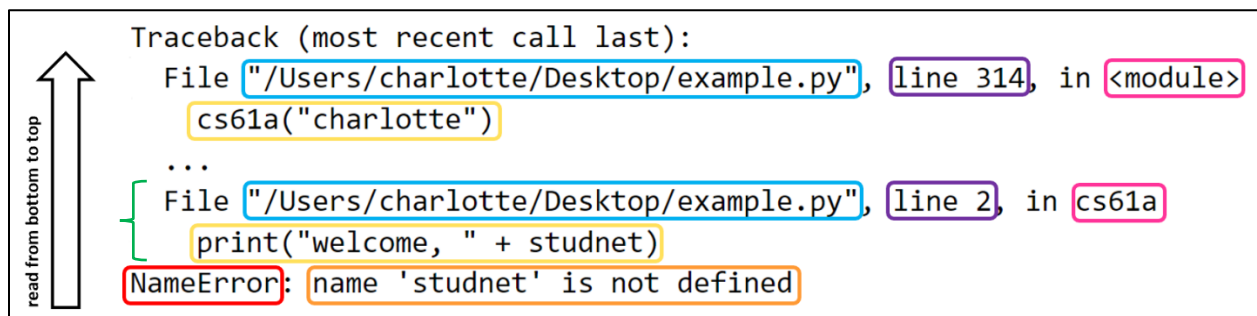
### Step 1: Open Terminal

- If running your code causes an error, your terminal should have a **traceback message**.
- To confirm, the output should start with `Traceback (most recent call last):`.
- Here is an example traceback message:

```
Traceback (most recent call last):
  File "/Users/charlotte/Desktop/example.py", line 314, in <module>
    cs61a("charlotte")
  ...
  File "/Users/charlotte/Desktop/example.py", line 2, in cs61a
    print("welcome, " + studnet)
NameError: name 'studnet' is not defined
```

### Step 2: Decomposition

- To interpret this output, we need to approach it by reading from **bottom to top**—this allows us to trace our error backward (traceback).
- We have broken down each section with **different colors**:



```
Traceback (most recent call last):
  File "/Users/charlotte/Desktop/example.py", line 314, in <module>
    cs61a("charlotte")
  ...
  File "/Users/charlotte/Desktop/example.py", line 2, in cs61a
    print("welcome, " + studnet)
NameError: name 'studnet' is not defined
```

read from bottom to top

### Step 3: Identify Error Type (Red Box)

- Firstly, you must identify the type of error you're encountering.
- Try to understand what the error means. You can always search it up!
- For definitions and examples of common error types, check out the [debugging guide](#).

### Step 4: Understand Error Message (Orange Box)

- This part of the traceback directly explains the error for you.
  - Once again, you can quickly search it up if you're not sure what it means.
  - For now, you're just trying to get an idea of the error.
  - You probably don't want to spend more than 1-2 minutes here.
  - However, you might need to come back to dig in more later.
- Sometimes, the name of the error is enough to correctly diagnose the issue.
- If not, let's begin tracing our error backward by moving up the traceback.

### Step 5: Investigate the Error Directly (Green Brace)

- In this step, *only* consider the two lines right above the last line.
- These two lines form a **line pairing** since they're grouped together.
- This line pairing gives us important direct information about the error:
  - Line Number (Purple Box)
    - This shows the line number in your code that caused the problem.
  - Code Executed (Yellow Box)
    - This shows you the line in your code that triggered the error.
    - Note: if there's a `SyntaxError`, you should see a `^` pointing directly to the part of the line that needs to be fixed.
  - Function Name (Pink Box)
    - This tells you the function that contains the error.
    - You may not have this if the error isn't in a function.
  - File Name (Blue Box)
    - This tells you the name of the file that's causing the error.
    - Moreover, it shows you the path to the file.

### Step 6: Analyze Information

- In **steps 3-4**, you identified the error **type & possible causes**.
- In **step 5**, you identified the error **location & context**.
- Now, ask yourself: *Does it make sense that this line in my code has this error?*
  - Yes

- Fantastic! Now that you understand exactly why your code doesn't work, you should start interacting with your code to find solutions.
- Our [debugging techniques](#) may be helpful. Test your changes often!
- No
  - If you're running your program correctly, the traceback message will always be accurate.
  - Try to understand every part of the line that is triggering an error and thoroughly think about ways it can potentially be faulty. Sometimes, it may help to take a break and step back so you can see the big picture.
  - Otherwise, please don't be afraid to reach out! You can ask questions/make a private post on Piazza or come to office hours!
- I Know There's Not Enough Information
  - The first line pairing doesn't always provide enough information.
  - For example, the error may be triggered by certain variables or arguments defined elsewhere.
  - Thus, we can investigate the previous functions calls!
  - You should only move onto the next step if you have a general idea of what information you're missing.

## Step 7: Backtrack with Function Calls

- Move up a single line in your traceback message.
- There should be three lines underneath the line that you're currently on.
- Starting from here and upward, each line pairing represents a **function call**.
- Remember: the previous function call will always be the line pairing directly above.
- We can investigate the function that made the last function call to see if it contains information we're missing. This is similar to step 5:
  - Line Number ([Purple Box](#))
    - This shows the line number in your code of the function.
  - Code Executed ([Yellow Box](#))
    - This shows you the actual line in your code where the function was called.
  - Function Name ([Pink Box](#))
    - This tells you where the current function was called.
    - Thus, this represents the parent function of the current function.
  - File Name ([Blue Box](#))
    - This shows you the name of the file that's making the function call.
    - Moreover, it shows you the path to it.
- You can keep moving up the traceback message if you still need more information!