Astro 142 Discussion 5

2023-10-17
Clea Hannahs

Last week recap

Object-oriented programming

Complex data types - arrays and structured arrays, custom dtype

Project 2 due Friday!

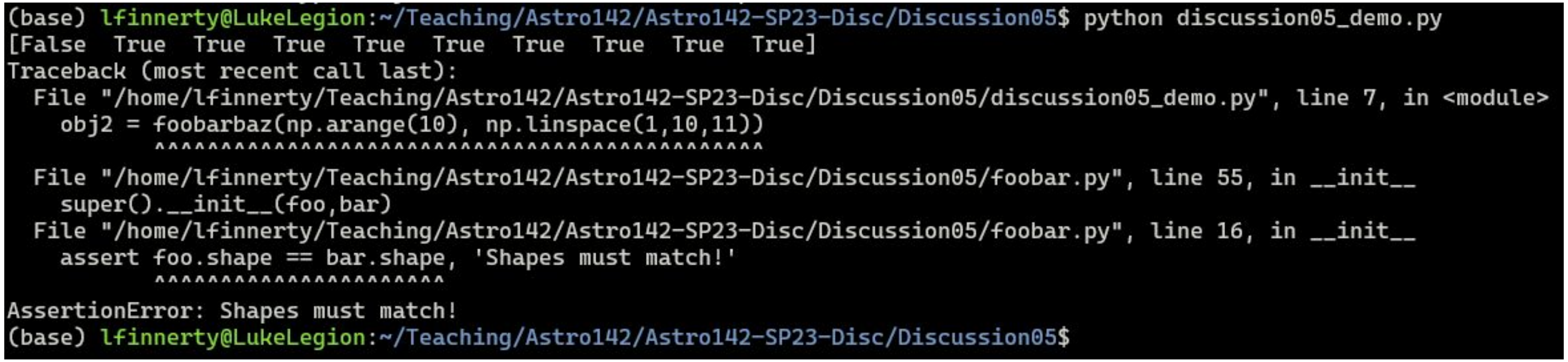
- Start early!
- Come to office hours!

Demo - Python traceback

Errors print tracebacks for debugging. Last line tells you the type of error, and any associated message

Last call is the line that actually raised the exception

Lines above tell you what the code was doing at the time (including line numbers), helping you “trace back” to the original source of the error



```
(base) lfinnerty@LukeLegion:~/Teaching/Astro142/Astro142-SP23-Disc/Discussion05$ python discussion05_demo.py
[False  True  True  True  True  True  True  True  True  True]
Traceback (most recent call last):
  File "/home/lfinnerty/Teaching/Astro142/Astro142-SP23-Disc/Discussion05/discussion05_demo.py", line 7, in <module>
    obj2 = foobarbaz(np.arange(10), np.linspace(1,10,11))
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/lfinnerty/Teaching/Astro142/Astro142-SP23-Disc/Discussion05/foobar.py", line 55, in __init__
    super().__init__(foo,bar)
  File "/home/lfinnerty/Teaching/Astro142/Astro142-SP23-Disc/Discussion05/foobar.py", line 16, in __init__
    assert foo.shape == bar.shape, 'Shapes must match!'
           ^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Shapes must match!
(base) lfinnerty@LukeLegion:~/Teaching/Astro142/Astro142-SP23-Disc/Discussion05$
```

Python inheritance

Tutorial: https://www.w3schools.com/python/python_inheritance.asp

See foobar.py in the Discussion05 folder for a basic example

super() allows calling methods from the parent class of the same name, useful for constructors

Can override parent class methods by defining a method with the same name in the child class

Non-overridden methods are accessible to instances of the child class

Note that numpy has a slightly different constructor naming convention (see HW2), same principles apply

Demo - Python assertions

Usage: `assert [conditional], [optional message string]`

See tutorial here: https://www.w3schools.com/python/ref_keyword_assert.asp

If conditional is true, code keeps going. If false, raises an `AssertionError` and prints the optional message string

Very useful for quick checks of inputs and for debugging

`assert 1==0` will stop the code at that line

Demo - Python exception handling

Tutorial: https://docs.python.org/3/tutorial/errors.html

We can tell python what to do when it encounters an error (exception) using a `try … except …` block

Can even specify doing different things depending on what error is encountered

`except: pass (or continue)` will ignore the error and keep going

Unhandled exceptions will crash your program and print the traceback!

Can raise your own exception using `raise [exception]`

Often we want to keep track of what went wrong - “catch” exceptions and write them to a log

Demo - python debugging with PDB

Assert and print statements are good for simple debugging

For more complicated programs, debuggers can be helpful

For python: `import pdb; pdb.set_trace()` or add `breakpoint()` anywhere in newer versions. Code will run until it hits `set_trace()` or `breakpoint()` and then enter the debugger, allowing you to step through line-by-line

Can also invoke from the command line: `python -m pdb myscript.py`

Demo - Python logger

Tutorial: https://docs.python.org/3/howto/logging.html

Documentation: https://docs.python.org/3/library/logging.html

Can log to an external file, making a record of what the program did

Can use log level system for more details when debugging, less detail (and associated overhead) when running normally

Useful for keeping track of handled exceptions

Short Answer Questions

Write Newton’s method in comments

See here: https://en.wikipedia.org/wiki/Newton's_method

```
# Define the function f(x) for which we want to find the root.
# Define f'(x), the derivative of f(x).

# Set an initial guess for the root, x_0.

# Set a tolerance level  for the accuracy of the approximation.

# Set a maximum number of iterations

# Initialize an iteration counter, iterations = 0.

# Start an iterative process:
# while iterations < max_iterations:
#     Calculate the next approximation using Newton's method
#     x_1 = x_0 - f(x_0) / f'(x_0)

#     Check if |x_1 - x_0| is less than the tolerance:
#     if |x_1 - x_0| < tolerance:
#         Break the loop; the approximation is accurate enough.

#     Update x_0 with the new approximation, x_0 = x_1.

#     Increment the iteration counter, iterations = iterations + 1.

# If the loop terminated due to reaching the maximum number of iterations, indicate that Newton's method did not converge.
# Otherwise, the approximation x_1 is the root we are looking for.
```

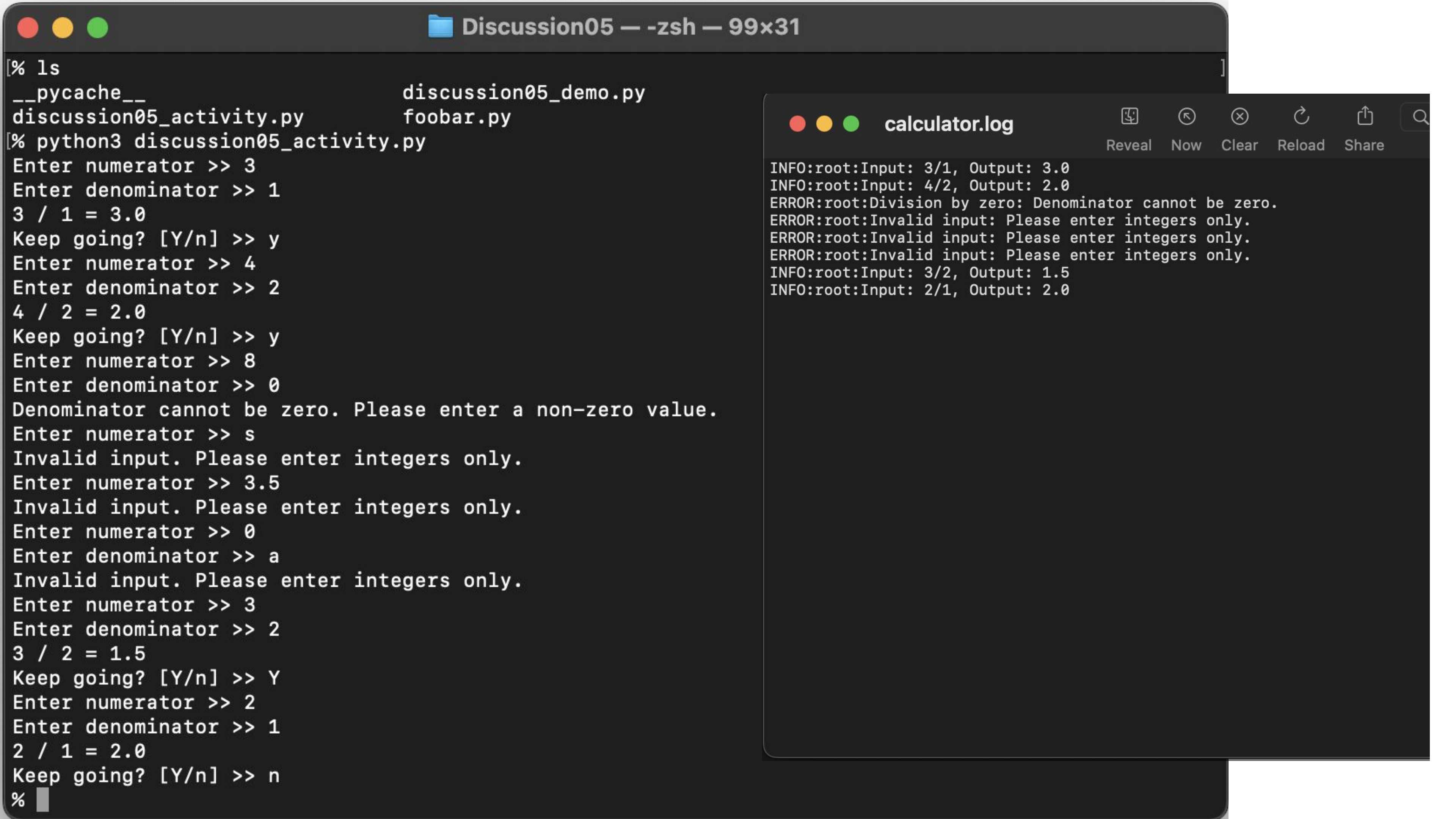
Discussion Activity

Discussion 5 Activity

Pull from the discussion github repo and go to the Discussion05 folder

Before you start modifying things, rename the _template.py file! This will minimize future merge conflicts

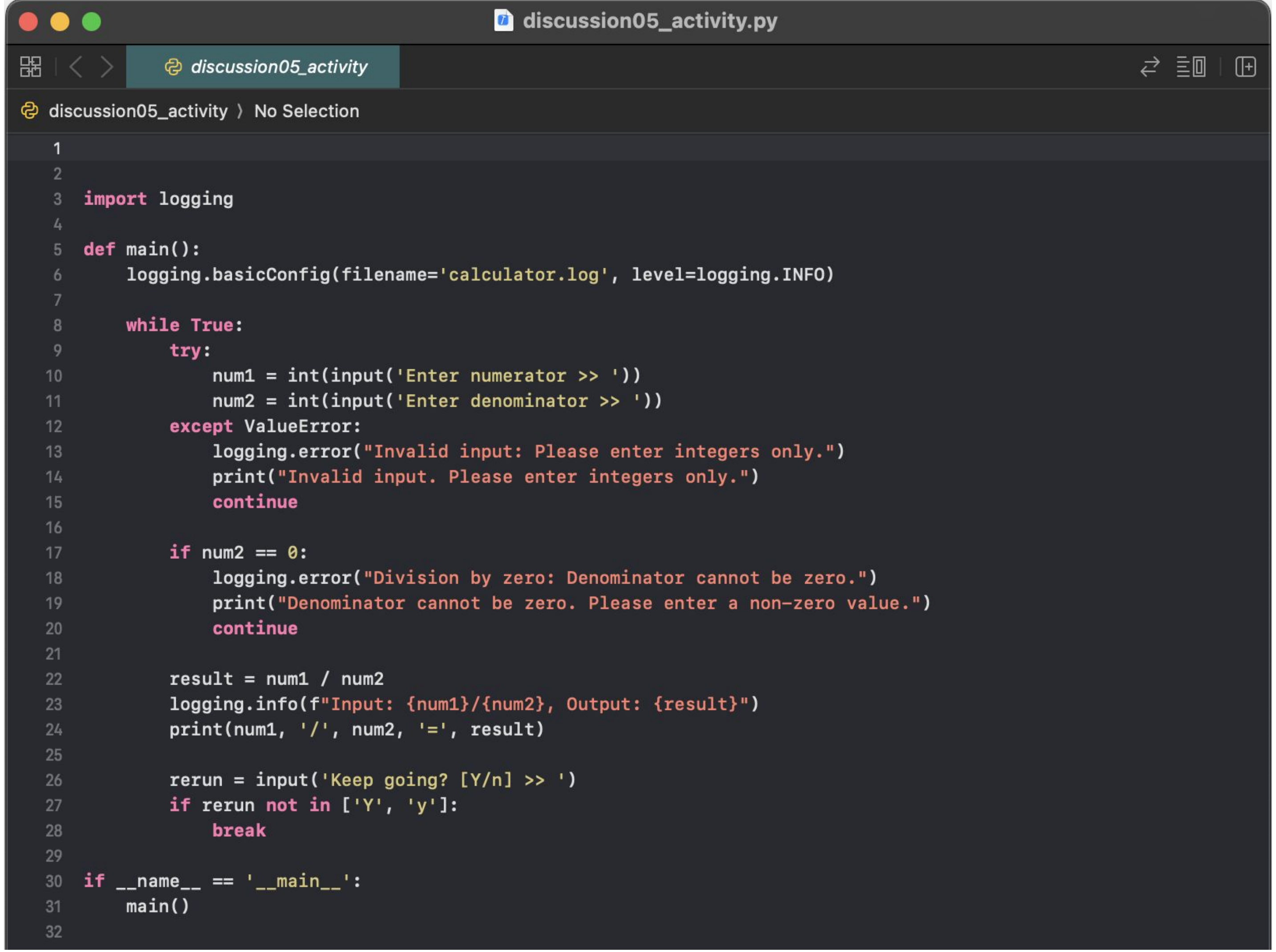
Paste a screenshot of your log and terminal output from some (handled) edge cases on the next slide


Discussion05 — -zsh — 99×31

```
% ls
__pycache__                      discussion05_demo.py
discussion05_activity.py         foobar.py
% python3 discussion05_activity.py
Enter numerator >> 3
Enter denominator >> 1
3 / 1 = 3.0
Keep going? [Y/n] >> y
Enter numerator >> 4
Enter denominator >> 2
4 / 2 = 2.0
Keep going? [Y/n] >> y
Enter numerator >> 8
Enter denominator >> 0
Denominator cannot be zero. Please enter a non-zero value.
Enter numerator >> s
Invalid input. Please enter integers only.
Enter numerator >> 3.5
Invalid input. Please enter integers only.
Enter numerator >> 0
Enter denominator >> a
Invalid input. Please enter integers only.
Enter numerator >> 3
Enter denominator >> 2
3 / 2 = 1.5
Keep going? [Y/n] >> Y
Enter numerator >> 2
Enter denominator >> 1
2 / 1 = 2.0
Keep going? [Y/n] >> n
%
```

calculator.log

```
INFO:root:Input: 3/1, Output: 3.0
INFO:root:Input: 4/2, Output: 2.0
ERROR:root:Division by zero: Denominator cannot be zero.
ERROR:root:Invalid input: Please enter integers only.
ERROR:root:Invalid input: Please enter integers only.
ERROR:root:Invalid input: Please enter integers only.
INFO:root:Input: 3/2, Output: 1.5
INFO:root:Input: 2/1, Output: 2.0
```



```python


import logging

def main():
    logging.basicConfig(filename='calculator.log', level=logging.INFO)

    while True:
        try:
            num1 = int(input('Enter numerator >> '))
            num2 = int(input('Enter denominator >> '))
        except ValueError:
            logging.error("Invalid input: Please enter integers only.")
            print("Invalid input. Please enter integers only.")
            continue

        if num2 == 0:
            logging.error("Division by zero: Denominator cannot be zero.")
            print("Denominator cannot be zero. Please enter a non-zero value.")
            continue

        result = num1 / num2
        logging.info(f"Input: {num1}/{num2}, Output: {result}")
        print(num1, '/', num2, '=', result)

        rerun = input('Keep going? [Y/n] >> ')
        if rerun not in ['Y', 'y']:
            break

if __name__ == '__main__':
    main()

```