327: Object-oriented programming

Lecture 7

9/22/2021

Professor Barron

Homework

- You can submit test inputs to the staff solution on gradescope
  - ungraded assignment named HW1 - staff solution
  - submit a file named “in.txt”
- HW2 given out next week after submissions for HW1 are fully closed
  - build on HW1 to include error handling and logging
  - solution for HW1 will be given

Today…

- Exceptions and error handling (Chapter 4)
- Logging

Coming up…

- Chapter 5, More OOP ideas and examples
- Some tidbits from Chapters 6-8
- Chapter 12, Testing
- C++ crash course
- Chapters 9-11 Design patterns

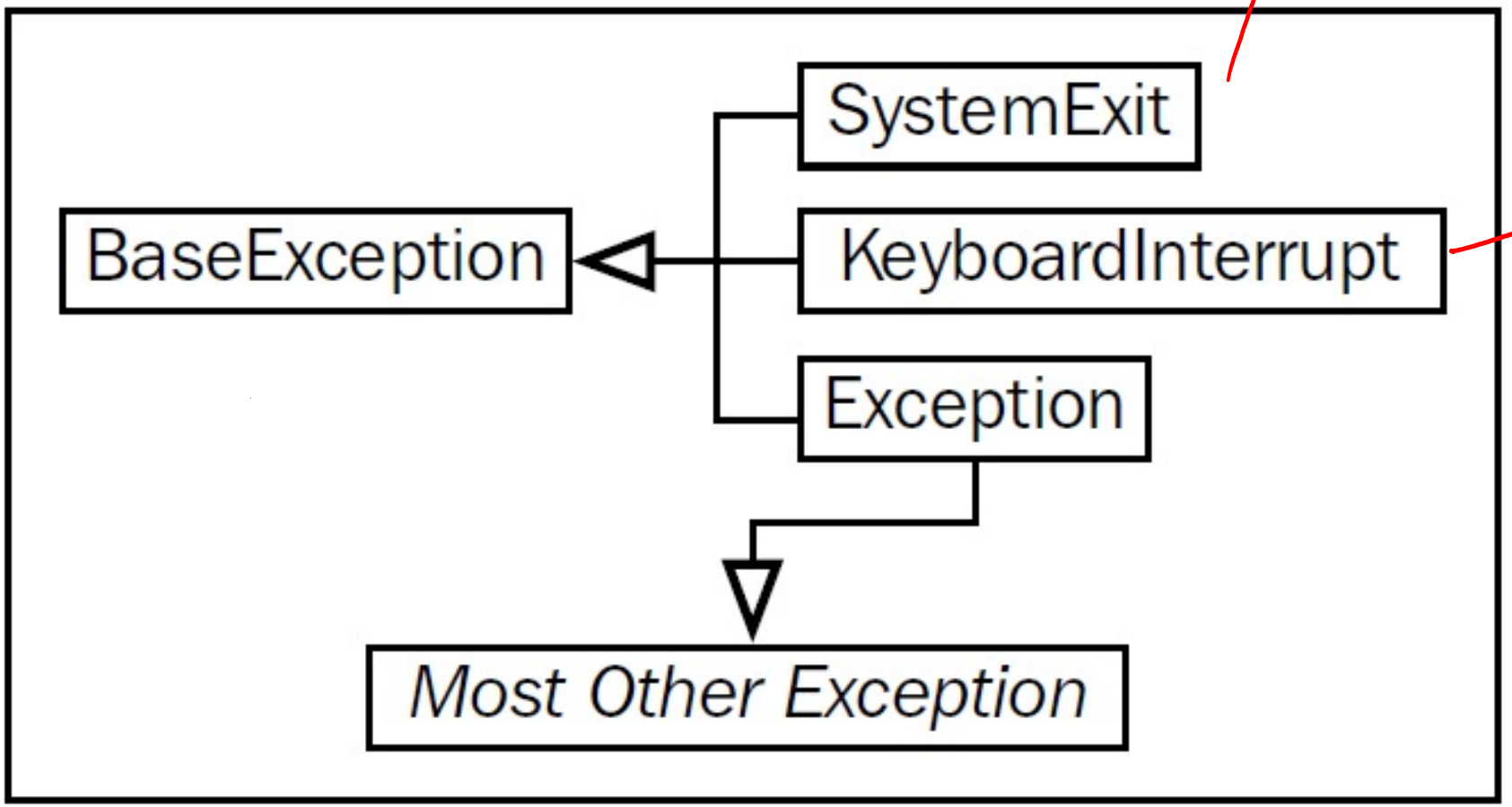
Error handling

- Check for exceptional cases
  - See how many questions on Ed for HW1…
- Example…
  - Check that dates are entered in the right format for new transactions
- Hard to write checks for every case
- Code becomes cluttered
- Checks have to run every time even if the error is rare

Exceptions and OOP

- Easier to ask forgiveness than permission (EAFP)
- When something goes wrong, then we handle it
- Exceptions are objects
- We may define our own classes of exceptions
- In a try… except block, exceptions are matched based on inheritance
- Sometimes used for deliberate control flow (not just errors)

Exception hierarchy



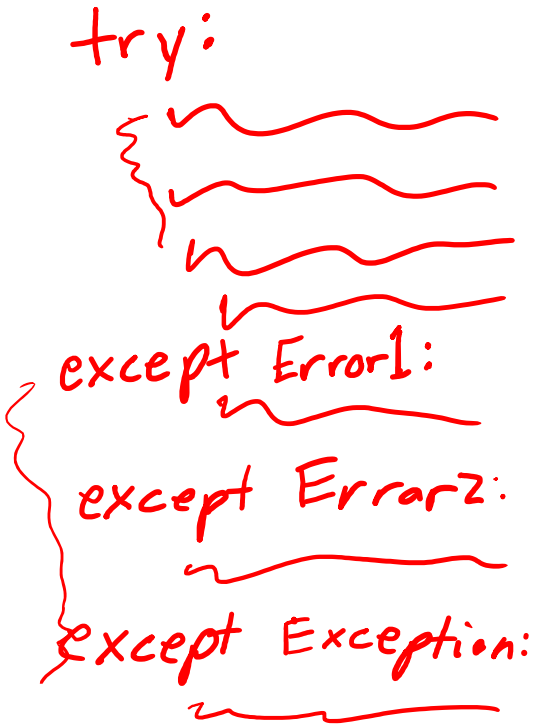
hitting ☒ in window corner
ctl+d
sys.exit(0)
ctl+c

What happens when an exception is raised?

1. Control flow is interrupted
2. If inside a try block, check excepts to see if they match the type
3. If handled, the program continues after the end of the try-except
    1. Finally clause happens regardless (even if the except block returned)
4. Otherwise, go up in scope. Functions exit and calling code is treated as having thrown the exception.
5. Repeat from 2
6. If exception reaches the top of the stack, program terminates and prints traceback

Exception tips

- If a built-in exception type fits, use it
  - More readable and re-usable
- Multiple small try blocks may be better than one large try with multiple except
  - Less likely to handle the wrong exception by accident
  - Remaining code in try is skipped when an exception is handled
- Sometimes it is okay to propagate an exception rather than handle it
  - The calling code may know how to handle it better



Exception tips

- Avoid suppressing exceptions (catching and doing nothing)
- Exceptions can be nested
  - not necessarily bad as long as it remains readable
  - similar to nested if/else

```
except TypeError:
    pass
```

```
try
    ...
except
    try
        ...
    except
        ...
```

```
try
    ...
    try
        ...
    except Error1
except Error2
```

Logging

- Valuable for error handling and debugging
- Caught an exception to keep the program running, but still want to know about it
- How can we do better than print()?
- Separate logs from primary output
- Filter by importance
- Easily change output stream, format, log level

Logging

singleton

```
1   import logging
2
3   # log messages to a file, ignoring anything less severe than ERROR
4   logging.basicConfig(filename='myprogram.log', level=logging.ERROR)
5
6   # these messages should appear in our file
7   logging.error("The washing machine is leaking!")
8   logging.critical("The house is on fire!")
9
10  # but these ones won't
11  logging.warning("We're almost out of milk.")
12  logging.info("It's sunny today.")
13  logging.debug("I had eggs for breakfast.")
14
15  try:
16      age = int(input("How old are you? "))
17  except ValueError as err:
18      logging.exception(err)
```

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0