PIC 16, Winter 2018

Lecture 3M: Exceptions

Monday, January 22, 2018

Matt Haberland

Announcements

- Assignment 2W due

Exceptions in C++ or Java

The following three slide show how I would introduce exceptions for C++ or Java

Errors

- Up until now, you've only seen *errors*
    - Compile-time errors – when the programmer breaks the rules of the language and the program can't run properly
    - Run-time errors – when the code executes in such a way that rules are broken (index out of bounds, out of memory, etc…) and the program can't continue
    - Logic errors – when the code executes completely, but did not achieve the desired result
- Usually, all of these are the programmer's fault.
    - These are "bugs".
    - They are not desirable.
    - They should not be present in finished code.
    - The program needs to be fixed.

Exceptions

- Exceptions are different
- They represent exceptional cases that *aren't* the programmer's fault
- They are probably not part of the *primary* flow of the program, but they cannot be prevented altogether.
- They may occur in finished code
- If so, they should be handled

Rough analogies

- Errors
  - Compile-time error: can’t start car because you don’t have the key
  - Run-time error: running out of gas
  - Logic error: driving to the wrong destination

These are typically preventable. You should *prevent* them if you want to your destination.

- Exceptions
  - Tire goes flat
  - Another car cuts you off
  - There’s a tree in the road

These aren’t preventable, but if you *handle* them properly, you can still get to your destination.

What can go wrong?

- Sending an email
  - Wrong email address (undeliverable)
  - Message too big (or inbox full, etc…)
  - Not connected to internet
- Solving an equation
  - Algorithm can’t find solution
  - Solution doesn’t exist
- User interacting with a GUI
  - User enters a string where numeric input is expected
  - Try to get/set element, but index is out of bounds
- What should methods that try to do these things return when something goes wrong?

What to do?

- Options:
    - Completely ignore it
    - Print a warning message
    - Return a “sentinel” value
    - Terminate with an assertion

No one of these options is *always* satisfactory.

- Alternative: return a custom object that contains:
        - usual return type for when everything works, and
        - an informative string for when something goes awry.

But this is inconvenient.

- We want a convenient, standard mechanism for dealing with things that go wrong

UCLA

Exceptions in Python

It’s not as clear-cut in Python.

Exceptions are more general

- In Python, exceptions are used to indicate other than when something went *wrong in the program*:
  - `KeyboardInterrupt`
  - `StopIteration`

`try-except` is used liberally

- In Python, it's not *just* to respond to situations you can't avoid
- BFAP vs LBYL

When to `raise` exceptions

- When something exceptional happens in your function or method (let’s call it `fun`) that your function can’t handle
- Whatever function, method, or script called your `fun` to make that occur will have to deal with the exception
- Get a better sense of how exceptions are used by looking into the exceptions that are raised by functions you use
  - Some functions are built into the language, so you can’t see how they are raised exactly
    - `1/0`
    - `enumerate(1)`
  - Others you can see the source code:
    - `import json; json.loads('not a valid json string')`

UCLA