Network Exceptions

Socket Operations Exceptions

- **OSError:** will be raised for nearly every failure that can happen at any stage in network transmission
- **socket.gaierror:**
  - This exception is raised when **getaddrinfo**() cannot find a name or service about which you.
  - Or if you supply a hostname instead of an IP address to a call like **bind**() or **connect**() and the hostname lookup fails

socket.gaierror Example

```
import socket

sok=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
try:

sok.connect(('nonexistent.hostname.foo.bar', 80))
except socket.gaierror as e:
    print(e.errno)
    print(e.strerror)
    raise
```

```
11001
getaddrinfo failed
```

Socket Operations Exceptions

- **socket.timeout:** This exception is raised only if you, or a library that you are using, decides to set a timeout on a socket rather than be willing to wait forever for a send() or recv() to complete. It indicates that the timeout indeed expired before the operation could complete normally

socket.timeout Example

```
import socket

serv_add=('127.0.0.1',12345)
sok=socket.socket(socket.AF_INE
T,socket.SOCK_STREAM)
sok.settimeout(10)
try:
    sok.connect(serv_add)
    sok.send("Hello".encode())
    msg=sok.recv(1024)
except socket.timeout as e:
    raise RuntimeError('message not
recived yet')
```

```
import socket

serv_add=('127.0.0.1',12345)
sok=socket.socket(socket.AF_INET,s
ocket.SOCK_STREAM)
sok.setsockopt(socket.SOL_SOCKE
T,socket.SO_REUSEADDR,1)
sok.bind(serv_add)
sok.listen()
while True:
    print('waiting for client')
    conn,add=sok.accept()
    msg=conn.recv(1024).decode()
    print(msg)
```

Delivering exceptions

- **There are two approaches to delivering exceptions**
  1. One option is not to handle network exceptions at all. They will then be visible for processing by the caller, who can catch or report them as they choose.
  2. The other approach is wrapping the network errors in an exception of your own.

     Custom exceptions also give you the opportunity to craft error messages that describe exactly what your library was trying to
     accomplish when it ran afoul of the network.

Custom Exceptions

```
class DestinationError(Exception):
    def __str__(self):
        return '%s: %s' % (self.args[0], self.__cause__.strerror)
    # ...
try:
    host = sock.connect(address)
except socket.error as e:
    raise DestinationError('Error connecting to destination') from e
```

Catching and Reporting Network Exceptions

- Two methods for catch exception
- **Granular exception handlers**
- The granular approach to exceptions is to wrap a try...except clause around every single network call that you ever make and print out a pithy error message in its place.
  - try:
    deliver_updated_keyfiles(...)
    except (socket.error, socket.gaierror) as e:

Blanket exception handlers

- **Blanket exception handlers**
- This involves stepping back from your code and identifying big regions that do specific things, like these:
  - “This whole routine is about connecting to the license server.”
  - “All of the socket operations in this function are fetching a response from the database.”
  - “This last part is all cleanup and shutdown code.”
- Then, at the very top level of your program, catch all the FatalError exceptions that you throw and print the error messages out there

  except:
  FatalError('cannot send replies: {}'.format(e))

Thank You