Non-Blocking Sockets in Python

Socket Execution Concurrency

- Sockets are blocking by default e.g., calling `my_socket.recv`

  will, by default, pause Python script execution until at least 1 byte or more are available.
- This makes it difficult to have execution concurrency, e.g., to
  - perform other tasks while handling network connections
  - handle multiple network connections
- Concurrency can, however, be achieved in various ways, e.g.,
  - Non-blocking sockets with native polling
  - Threading
  - select
  - ...

Non-blocking Sockets with Native Polling

Blocking vs. Non-blocking Sockets

- TCP sockets are "blocking" by default, e.g., `recv` will not return until at least one byte of data is received. The same happens during `send/sendall`, i.e., the connection will block until the operation completes.
- e.g., EchoClientServer.py server blocks on "accept" and server/client blocks on recv.
- Blocking can lead to situations where code execution is prevented from performing other things while waiting.
- A socket can instead be placed in "non-blocking" mode, i.e.,

  `my_socket.setblocking(False)`

  This means that your code does not have to wait for the operation to complete. This is useful when, for example, you have multiple socket connections, and want to prevent one from blocking access to the others. Or, if you want to do something else instead of blocking.

Blocking vs. Non-blocking Sockets

- e.g., calling `recv` in non-blocking mode, will return data currently in the systems receive buffer for that socket, as usual.
- Unlike blocking mode, however, it will not wait for data to arrive. If the buffer is empty, it immediately generates a socket.error exception.
- The socket error exception can be caught and execution will continue.

Non-blocking Socket Example

- Say we want to check for incoming connections on a listening socket instance. We can use:

```
my_socket.setblocking(False)

try:
    client = my_socket.accept()
    # If the accept is successful, execution will immediately
    # resume here:
    ...
except socket.error:
    # If there is nothing to "accept()", then a socket.error
    # exception will occur. Execution will immediately resume
    # here:
    ...
```

See echo_polling_multiclient.py