This course is intended to teach programmers how to write server side programs that will be secure and maintainable. The initial section will write a simple web server, which is intended to demonstrate a number of potential weaknesses, as well as providing some understanding of the requirements of a web server. (For most of the later work, the Apache web server will be used, with IIS where appropriate, but that is beyond the time constraints for today). The course is aimed at those new to rexx, with some programming experience, but with some additional material could be used for a novice.

The programs are intendet to let students find mistakes and pitfalls, therefore learning from errors. This means that they will be used to making frequent incremental changes. There will be obvious errors in the pogic, but any typo's or bugs are usually mine; as you can see, I sometimes get some assistance.

As this is being heavily compressed to fit the presentation, I will be using the Blue Peter method, "here's one I created earlier". During a normal presentation, the students are intended to help develop the code, and discuss the dangers during development, therefore the order in which the problems are closed will change. The code will be available for download from www.protts.uk.eu.org. Don't use the code as a live web server!

We start with the traditional helloworld program (helloworld.rexx), mainly to verify the rexx processor is working. As soon as that works, we then move to some sockets programming to create a single use 'Hello World' server (helloworld1.rexx). This introduces the simple but commonplace sockets communication used by the main web protocols.

At this stage we want to make the server run for a bit longer, so we will start to use a loop, and introduce a function for producing the data. The data returned will be simple to start with, so we don't have to start with file handling yet (sw.rexx).

We started with a hard coded port, but this will be a problem for many developers, so we then introduce the idea of parameters and allow a port to be specified (sw1.rexx).

Next we are going to provide some real files, to make this a web server that works. Our getdata function is updated, but of course the main section of code is not changed (sw2.rexx).

By using real files, we will quickly see there is a problem, the css file is being returned as text/html, and Firefox doesn't like this (in fact it should accept as the request was specified Accept: text/css,*/*;q=0.1), there is a similar problem with images, although these are displayed regardless. At this stage the RFC will be looked at in detail, but again we don't have time today. We will fix the problem with a quick select on filetype (sw3.rexx).

So far we have blocked on each call, but we don't have to so let's introduce a simple non blocking technique with select (sw4.rexx).

At this stage we can stop the receive from blocking to avoid a basic denial of service, and demonstrate the problem using telnet (sw5.rexx).
The next phase is to become multi threaded, and we will use the rexx class handling for this. The change from procedural to object oriented is not too onorous at this stage, but we will take three steps, the last demonstrating early reply (sw6.rexx, sw7.rexx, sw8.rexx).

Early reply can be tricky if allowed too early, and the initial attempt probably was, so we look at how this can be improved (sw9.rexx).

We've been short on error checking so far, so time to work on that (swa.rexx).

Also so far our class structure has not been carefully designed, so a bit of work here (swb.rexx).

It may become irritating to have to mess around to kill the server, so now we will allow the console to become useful, at least allowing a clean close (swc.rexx).

We will work a bit more on the class structure, referring to the RFC to identify how we can simplify maintenance (swd.rexx, swe.rexx)

With the code a bit cleaner, we can handle larger files by reading in chunks (swf.rexx).

Of course a client can make multiple requests on a connection, so we can support this (swg.rexx, swh.rexx).

The last stage for today is a bit of clean up to make the headers closer to honouring the RFC, and also to put in some useful diagnostics (swi.rexx, swj.rexx).