**General Questions:**

**Why does Functional Programming matter? When should a functional programming language be used?**

*Functional languages use a different paradigm than imperative and object-oriented languages. They use side-effect-free functions as a basic building block in the language. ... So most people never got more out of functional programming than calculating a single output from a single input.*

**How do companies like Microsoft, Google, Opera and Mozilla profit from their browsers?**

<https://www.quora.com/How-do-companies-like-Microsoft-Google-and-Mozilla-profit-from-their-browsers>

**Why does opening a TCP socket have a large overhead?**

*9*

*I believe, generally speaking, that opening a TCP connection is considered expensive when compared to the ability to reuse already open connections by keeping it open. You are correct, opening a connecting will take only 3 packets/turns, but that time - 3 x your RTT - is far beyond the cost of reusing an already open connection, which is far closer to 0. The disparity grows even faster if you're opening and closing connections frequently.*

*You are certainly correct though, when compared to the number of turns you're going to see as the application "does it's thing," those 3 packets can seem pretty small, but again, it depends on how you want to compare the options AND how your application behaves/how many times you plan on opening a connection.*

*Edit If we're talking UDP vs. TCP though, Cheekaleek here is 100% correct - the overhead of is massive in the long term when compared to the connectionless operations of UDP*

**What is Encapsulation important for?**

*Encapsulation - The key advantage of using an Object Oriented Programming language like Java is that it provides your code - security, flexibility and its easy maintainability through encapsulation. ... Encapsulation is also useful in hiding the data(instance variables) of a class from an illegal direct access.*

**What is a real-time system and how is it different from an ordinary system?**

*Real time OS*

*1. A real-time operating system is an operating system intended to serve real-time applications that process data as it comes in, typically without buffer delays.*

*2. It is deterministic.*

*3. It is time sensitive.*

*4. It can’t use virtual memory.*

*5. It is dedicated to single work.*

*6. It has flat memory model.*

*7. It has low interrupt latency.*

*Non-real time OS*

*1. A Non-real time OS or General purpose OS is the operating system made for high end, general purpose systems like a personal computer, a work station, a server system etc.*

*2. It is not deterministic.*

*3. It is time insensitive.*

*4. It can use virtual memory concept.*

*5. It is used in multi-user environment.*

*6. It has protected memory model.*

*7. It has high interrupt latency*

**What's the relationship between real-time languages and heap memory allocation?**

*Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM . Variables allocated on the stack are stored directly to the memory and access to this memory is very fast, and it's allocation is dealt with when the program is compiled.*

*https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap*

**Immutability is the practice of setting values once, at the moment of their**

**creation, and never changing them. How can immutability help write safer code?**

*Both mutable and immutable objects have their own uses, pros and cons.*

*Immutable objects do indeed make life simpler in many cases. They are especially applicable for value types, where objects don't have an identity so they can be easily replaced. And they can make concurrent programming way safer and cleaner (most of the notoriously hard to find concurrency bugs are ultimately caused by mutable state shared between threads). However, for large and/or complex objects, creating a new copy of the object for every single change can be very costly and/or tedious. And for objects with a distinct identity, changing an existing objects is much more simple and intuitive than creating a new, modified copy of it.*

*Think about a game character. In games, speed is top priority, so representing your game characters with mutable objects will most likely make your game run significantly faster than an alternative implementation where a new copy of the game character is spawned for every little change.*

*Moreover, our perception of the real world is inevitably based on mutable objects. When you fill up your car with fuel at the gas station, you perceive it as the same object all along (i.e. its identity is maintained while its state is changing) - not as if the old car with an empty tank got replaced with consecutive new car instances having their tank gradually more and more full. So whenever we are modeling some real-world domain in a program, it is usually more straightforward and easier to implement the domain model using mutable objects to represent real-world entities.*

*Apart from all these legitimate reasons, alas, the most probable cause why people keep creating mutable objects is inertia of mind, a.k.a. resistance to change. Note that most developers of today have been trained well before immutability (and the containing paradigm, functional programming) became "trendy" in their sphere of influence, and don't keep their knowledge up to date about new tools and methods of our trade - in fact, many of us humans positively resist new ideas and processes. "I have been programming like this for nn years and I don't care about the latest stupid fads!"*

**What are the pros and cons of mutable and immutable values.**

*Many functional languages are non pure (allow mutation and side effects).*

*f# is for example, and if you look at some of the very low level constructs in the collections you'll find that several use iteration under the hood and quite a few use some mutable state (if you want to take the first n elements of a sequence it's so much easier to have a counter for example).*

*The trick is that this is something to generally:*

*Use sparingly*

*Draw attention to when you do*

*note how in f# you must declare something to be mutable*

*That it is possible to largely avoid mutating state is evidenced by the large amount of functional code out there. For people brought up on imperative languages this is somewhat hard to get your head round, especially writing code previously in loops as recursive functions. Even trickier is then writing them, where possible, as tail recursive. Knowing how to do this is beneficial and can lead to far more expressive solutions that focus on the logic rather than the implementation. Good examples are those that deal with collections where the 'base cases' of having no, one or many elements are cleanly expressed rather than being part of the loop logic.*

*It is really 2 though that things are better. And this is best done via an example:*

*Take your code base and change every instance variable to readonly[1][2]. Change back only those ones where you need them to be mutable for your code to function (if you only set them once outside the constructor consider trying to make them arguments to the constructor rather than mutable via something like a property.*

*There are some code bases this will not work well with, gui/widget heavy code and some libraries (notably mutable collections) for example but I would say that most reasonable code will allow over 50% of all instance fields to be made readonly.*

*At this point you must ask yourself, "why is mutable the default?". Mutable fields are in fact a complex aspect of your program as their interactions, even in a single threaded world, have far more scope for differing behaviour; as such they are best highlighted and drawn to the attention of the coder rather than left 'naked' to the ravages of the world.*

*It is notable that most functional languages have either no concept of null or make it very hard to use because they work, not with variables, but with named values whose value is defined at the same time (well scope) the name is.*

*I find it unfortunate that c# did not copy java's concept of immutability with local variables too. Being able to assert emphatically that something doesn't change helps make intent clear whether a value is on the stack or in an object/struct.*

*If you have NDepend then you can find these with WARN IF Count > 0 IN SELECT FIELDS WHERE IsImmutable AND !IsInitOnly*

**What's the Object-Relational impedance mismatch?**

*The object-relational impedance mismatch is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being served by an application program (or multiple application programs) written in an object-oriented programming language or style, …*

**Which principles would you apply to define the size of a cache?**

[*https://www.quora.com/What-are-the-most-important-factors-in-CPU-cache-design*](https://www.quora.com/What-are-the-most-important-factors-in-CPU-cache-design)

**What's the difference between TCP and HTTP?**

*HTTP is a protocol used mostly for browsing the internet (IE, Firefox, etc). It rides on top of TCP which provides a reliable link between two computers (if packet get lost - it is re-transmitted). TCP itself rides on top of IP, which provides unified addressing to communicate between computers. TCP/IP is a basis for internet and 99% of other networks.*

*Basically it means if you are communicating HTTP, you are doing it with TCP/IP underneath (but I am sure this is not what your professor meant).*

*Arduino Nano is not supporting all of those, so you need something in between, which will translate Nano signalling to TCP/HTTP communication.*

*Some of you options are:*

*Communicating with Nano over Serial and making PC translate your Serial protocol to HTTP/TCP*

*Switch Nano with some other Arduino board which supports Ethernet/Wifi shield extension (Uno/Mega), or choosing a custom board which contains Ethernet by itself*

*Using another Arduino (Uno/Mega) with Ethernet shield as an additional board which communicates with Nano over Serial or with the help of RF modules (I personally implemented this option in past)*

*Another unusual option is to attach Nano to your Android smartphone using Audio cable and to use soft-modem library (https://code.google.com/p/arms22/issues/detail?id=2), which contains implementation for Android and write an application for Android*

*Web server you mention supports HTTP only by definition, so if you want to communicate over TCP, you will need to use some TCP server.*

*One of the existing web services to provide graphs for visualizing Sensor data is https://xively.com/, it API is based on REST which rides on top of HTTP. But it is not the only one.*

**What are the tradeoffs of client-side rendering vs. server-side rendering?**

*Your content is visible to search engines like Google.*

*The page loads faster. There's no "white page" while the browser downloads the rendering code and data and runs the code.*

*It maintains the idea that pages are documents, and if you ask a server for a document by URL, you get back the text of the document rather than a program that generates that text using a complicated API.*

*Read More*

[*https://www.quora.com/What-are-the-tradeoffs-of-client-side-rendering-vs-server-side-rendering*](https://www.quora.com/What-are-the-tradeoffs-of-client-side-rendering-vs-server-side-rendering)

**How could you develop a reliable communication protocol based on a non-reliable one?**

[*http://www.gerhardmueller.de/docs/UnixCommunicationFacilities/ip/node11.html*](http://www.gerhardmueller.de/docs/UnixCommunicationFacilities/ip/node11.html)

**Tony Hoare who invented the null reference once said "I call it my billion-dollar mistake" since it lead to "innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years". Imagine you want to remove the possibility to have null references in your preferred language: how would you achieve this goal? What consequences could this have?**

*null is evil*

*There is a presentation on InfoQ on this topic: Null References: The Billion Dollar Mistake by Tony Hoare*

*Option type*

*The alternative from functional programming is using an Option type, that can contain SOME value or NONE.*

*A good article The “Option” Pattern that discuss the Option type and provide an implementation of it for Java.*

*I have also found a bug-report for Java about this issue: Add Nice Option types to Java to prevent NullPointerExceptions. The requested feature was introduced in Java 8.*