0:14

Language is an interesting idea. The word language is used to describe a system that we use to communicate our thoughts and ideas with each other. There are different ways in which the system can be used. We can communicate through writing, reading, speaking, drawing pictures or even making gestures with our body. Language has allowed us to create incredible things like the pyramids and solve complex problems that help us understand our world. It even helps us communicate the most basic things to other people. But, as with many things in our lives, we have to adapt language to meet our needs. Language has evolved over the many years of human existence. Think about how language has changed with the integration of modern technology and the rise of Internet culture. Programming languages evolved in a similar fashion as traditional languages. Each new programming language was developed to provide solutions to problems that previous languages were unable to adequately address. Over the years, ideas used in computer languages have caused a shift in programming paradigms. We needed to change the way in which programs were written in order to address the latest problems more effectively. If you're a veteran of the software industry, you may remember languages such as COBOL and Fortran. If you're new to the software scene, these are languages you may have heard of but have never actually used. So, what programming paradigm do these two languages follow? Have you met Ted? Ted is a software developer making his way through the different ages of programming languages and programming paradigms. We'll be following Ted from the early years of computers, when computers were designed to simply process batches of input into output. Ted will finish his journey in our modern day society, where computers help us with complex tasks. Ted begins his career in the 1960s. He has just been hired by a bank to develop their very first program that will be used to keep track of account balances. In these days, the two most popular programming languages were COBOL and Fortran. They followed an imperative paradigm which broke up large programs into smaller programs called subroutines, which are like methods in Java. Now back in the 1960s, computer processing time was costly. As a result, it was important to maximize processing performance. This was accomplished by having global data because they are all located in one place in the computer's memory for a program. With globally accessible variables, all the subroutines would be able to access them to do their necessary calculations. However, there are some problems that Ted noticed as he continued with his career. With global data, it was possible that changes in the data could have weird side effects on the program. Sometimes, a subroutine would run into cases where the global data was not as expected. The need for better data management led to changes to imperative programming and the rise of languages like Algol 68 and Pascal in the 1970s. The idea of local variables was introduced. Subroutines were called procedures which could contain nested procedures. And each one could have their own variables. Algol 68 and Pascal support the notion of an abstract data type, which is a datatype that is defined by the programmer and not built into the language. An abstract data type is essentially a grouping of related information that is denoted with a type. It was a way to organize data in a meaningful way. Developers can write their software using these types in a similar way to the built-in types of the languages. By having variables in different scopes, Ted can compartmentalize the data into different procedures. This way, a procedure can be the only one that can modify that piece of data, allowing Ted to put it in the local scope and not have to worry about it being changed by another procedure. As we continue on our journey into the mid-1970s, computer processing time became less expensive while human labor became more expensive. The time consuming factor in software development was now the human element. Problems were becoming more complex. The questions we could ask computers to solve were becoming more intricate. For developers like Ted, this meant that software was becoming so massive that having one file for his program was becoming difficult to maintain. New languages arose such as C and Modula-2 that provided a means to organize programs and allow developers to more easily create multiple but unique copies of their abstract data types. Programs could now be organized into separate files. In C, each file contained all the associated data and functions that manipulated it and it declared what could be accessed through a separate file called the Header File. There are still issues that are not addressed in any of the languages we've looked at so far. These languages do not make it easy for an abstract data type to inherit from another. That means that Ted can define as many data types as he wants but cannot declare that one type is an extension of another type. In the final leg of our journey, let's follow Ted into the 1980s. During this time in software history, the concepts of Object-Oriented Design, that are central for object-oriented programming, became popular. The goal of object-oriented design is to make an abstract data type easier to write, structure a system around abstract data types called classes and introduce the ability for an abstract data type to extend another by introducing a concept called inheritance. With an object-oriented programming paradigm, Ted is now able to build a software system that is made up of entirely abstract data types. The advantage of this is that the system will mimic the structure of the problem, meaning that any object-oriented program is capable of representing real world objects or ideas with more fidelity. Class files replace the standard files in C and Modula-2. Each class defines a type with associated data and functions. These functions are also known as methods. A class acts like a factory, making individual objects, all of a specific type. This allows Ted to compartmentalize the data and how it can be manipulated in their own separate classes. Object-Oriented Programming is the predominant programming paradigm. Popular modern languages such as Java, C++ and C# are all founded based on objects. So, why the big history lesson? Why do we care about Ted and his journey through the different software design eras? It is important because as a software developer, you need to have a broad understanding of what is out there in the industry

today. There are many systems that still use the older languages and design paradigms. It is also important to understand that while object-oriented programming is a powerful tool, it is not the only one in your toolbox. Object-oriented design is not always the best approach for everything because the design may not fit the problem. Remember, that it is more important to be efficient with your time even if this means taking a non-object-oriented approach. As the power of computers becomes better than ever before, the most expensive cost of creating software is you.