Chapter 1 : Computer science - Wikipedia

Java is an Object-Oriented programming language. Since Object-Oriented programming (OO) is currently one of the most popular programming paradigms, you will need to learn its fundamental concepts in order to build a career in Computer Science.

High School Statutory Authority: Students shall be awarded one-half to one credit for successful completion of this course. The prerequisite for this course is proficiency in the knowledge and skills relating to Technology Applications, Grades This course is recommended for students in Grades Students will learn about the computing tools that are used every day. Students will foster their creativity and innovation through opportunities to design, implement, and present solutions to real-world problems. Students will collaborate and use computer science concepts to access, analyze, and evaluate information needed to solve problems. Students will learn the problem-solving and reasoning skills that are the foundation of computer science. By using computer science knowledge and skills that support the work of individuals and groups in solving problems, students will select the technology appropriate for the task, synthesize knowledge, create solutions, and evaluate the results. Students will learn digital citizenship by researching current laws and regulations and by practicing integrity and respect. Students will gain an understanding of the principles of computer science through the study of technology operations and concepts. The student develops products and generates new understanding by extending existing knowledge. The student is expected to: The student communicates and collaborates with peers to contribute to his or her own learning and the learning of others. The student locates, analyzes, processes, and organizes data. The student uses appropriate strategies to analyze problems and design algorithms. The student explores and understands safety, legal, cultural, and societal issues relating to the use of technology and information. The student understands technology concepts, systems, and operations as they apply to computer science. The required prerequisite for this course is Algebra I. Students will collaborate with one another, their instructor, and various electronic communities to solve the problems presented throughout the course. Through data analysis, students will identify task requirements, plan search strategies, and use computer science concepts to access, analyze, and evaluate information needed to solve problems. Students will gain an understanding of the principles of computer science through the study of technology operations, systems, and concepts. The student develops products and generates new understandings by extending existing knowledge. Students shall be awarded one credit for successful completion of this course. This course is recommended for students in Grades 11 and Students will gain an understanding of computer science through the study of technology operations, systems, and concepts. Students will gain an understanding of advanced computer science data structures through the study of technology operations, systems, and concepts. Students will collaborate to develop forensic techniques to assist with computer security incident response. Students will learn methods to identify, collect, examine, and analyze data while preserving the integrity of the information and maintaining a strict chain of custody for data. Students will solve problems as they study the application of science to the law. Students will gain an understanding of computing and networking systems that transmit or store electronic data. The required prerequisite for this course is Algebra II. Exposure to the mathematical concepts and discrete structures presented in this course is essential in order to provide an adequate foundation for further study. Discrete Mathematics for Computer Science is generally listed as a core requirement for Computer Science majors. Course topics are divided into six areas: Students will develop the ability to see computational problems from a mathematical perspective. Introduced to a formal system propositional and predicate logic upon which mathematical reasoning is based, students will acquire the necessary knowledge to read and construct mathematical arguments proofs , understand mathematical statements theorems , and use mathematical problem-solving tools and strategies. Students will be introduced to discrete data structures such as sets, discrete functions, and relations and graphs and trees. Students will also be introduced to discrete probability and expectations. Students will collaborate with one another, their instructor, and various electronic communities to solve gaming problems. Through data analysis, students will include the identification of task requirements, plan search strategies, and use programming concepts to access, analyze, and evaluate information needed to design games. By acquiring programming knowledge and skills that support the work of individuals and groups in solving problems, students will select the technology appropriate for the task, synthesize knowledge, create solutions, and evaluate the results. Students will create a computer game that is presented to an evaluation panel. The student understands technology concepts, systems, and operations as they apply to game programming. The required prerequisites for this course are proficiency in the knowledge and skills relating to Technology Applications, Grades , and Algebra I. Students will collaborate with one another, their instructor, and various electronic communities to solve problems presented throughout the course. Through data analysis, students will identify task requirements, plan search strategies, and use software development concepts to access, analyze, and evaluate information needed to program mobile devices. By using software design knowledge and skills that support the work of individuals and groups in solving problems, students will select the technology appropriate for the task, synthesize knowledge, create solutions, and evaluate the results. Students will gain an understanding of the principles of mobile application development through the study of development platforms, programming languages, and software design standards. The placement of the process standards at the beginning of the knowledge and skills listed for each grade and course is intentional. The process standards weave the other knowledge and skills together so that students may be successful problem solvers and use mathematics efficiently and effectively in daily life. The process standards are integrated at every grade level and course. When possible, students will apply mathematics to problems arising in everyday life, society, and the workplace. Students will use a problem-solving model that incorporates analyzing given information, formulating a plan or strategy, determining a solution, justifying the solution, and evaluating the problem-solving process and the reasonableness of the solution. Students will select appropriate tools such as real objects, manipulatives, paper and pencil, and technology and techniques such as mental math, estimation, and number sense to solve problems. Students will effectively communicate mathematical ideas, reasoning, and their implications using multiple representations such as symbols, diagrams, graphs, and language. Students will use mathematical relationships to generate solutions and make connections and predictions. Students will analyze mathematical relationships to connect and communicate mathematical ideas. Students will display, explain, or justify mathematical ideas and arguments using precise mathematical language in written or oral communication. Students will collaborate with one another, their instructor, and various electronic communities to solve problems in designing and programming robots. Through data analysis, students will identify task requirements, plan search strategies, and use robotic concepts to access, analyze, and evaluate information needed to solve problems. By using robotic knowledge and skills that support the work of individuals and groups in solving problems, students will select the technology appropriate for the task, synthesize knowledge, create solutions, and evaluate the results. Students will gain an understanding of the principles of robotics through the study of physics, robotics, automation, and engineering design concepts. The student uses mathematical processes to acquire and demonstrate mathematical understanding.

Chapter 2 : Department of Computer Science

Computer science is the theory, experimentation, and engineering that form the basis for the design and use of calendrierdelascience.com involves the study of algorithms that process, store, and communicate digital information.

View Larger Image If you aspire to be a software engineer and you are just starting out, you probably have asked yourself these questions: Because of all the noise out there, aspiring software engineers find it very hard to weed out the valuable information from the noise. So I decided to write an article that reflects my personal opinions and experiences. What I want to do here is to give you a very broad overview of how your CS career looks like from start to finish, what skills you must absolutely acquire, and what to expect at every step of the way. In doing so, I actually agree with many aspects presented in the mentioned article but I also disagree with some. The overall thesis of this article is that software engineers pass through three different phases. I am going to explain to you what these phases are. Afterwards, I am going to tell you exactly what skills you need to move from one phase to another. The three phases of a software engineer Any successful software engineer goes through three consecutive phases. These three phases are: Let me explain each one of these phases. The Coder Any software engineer starts his career as a coder. This can happen at a very young age. So what is a coder? A coder is someone who knows how to talk the language of a machine. Given a particular problem, a coder knows how to break down the problem into instructions that the machine can understand in order to come up with a solution. Here is the thing, if you find yourself really struggling at this phase, then you might want to consider a different career path because the coding phase is literally the easiest phase of your CS career. If you succeed, congratulations. You might have a successful career as a software engineer. Unfortunately many software engineers remain in this phase for their whole career. If you remain just a coder, your promotions will be severely limited. You need at least to evolve to the next phase for this to happen. You need to be a programmer. The Programmer Once you learnt the basics of at least two programming languages preferably one statically-typed and one dynamically-typed , you are a solid coder. The question now is how to promote yourself to the programmer status? A programmer is essentially a sophisticated coder. Writing code that does the job is what coders do but writing efficient code that does the job is what programmers do. Here is a list of some skills that you should have as a programmer: Now I got good news and bad news for you. I wrote an in-depth article that discusses everything you need to know about the coding interview process. The vast majority of software engineers retire at this phase. The Computer Scientist Learning does not stop after mastering the programming phase. As a matter of fact it actually starts! You have a solid understanding of designing large distributed systems and you know how to build scalable systems that can handle large loads and tolerate failures. A computer scientist also never stops learning, and always tries to stay up to date with the latest in technology. You might also need to cooperate with other teams. All of these require stellar social and leadership skills. In the rest of this article, I will go through the technical skills that you need in order to be a coder, then a programmer, and finally a computer scientist. This is the easiest step in your CS career, and it gives you a quick feedback about whether you should pursue a CS career or not. The reason is, at this stage what matters is not the particular programming language, but the concepts that you will be learning. These concepts will hold in almost any other programming language. Python I highly suggest you start with Python. Because Python is a language that is very easy to learn, like, really really easy! It is a very high-level language which allows you to write real programs in just a few lines of code. To learn python, I highly recommend Python Crash Course. I find this book to be very useful for beginners. I also like the fact that the book is project based so you will have fun building things while you are learning to code. Java Why another language though? The reason I recommend another language, especially Java, is because it will teach you some programming concepts that do not even exist in python. A combination of Python and Java is a very good way to start because together they provide you with a very solid idea of the programming concepts that you will need in almost any other programming language. To add to the benefits mentioned above, both python and java are heavily used in industry. So not only will you be spending your time learning the foundations that will pave the way for you to progress further, but also you will be learning some practical languages that are very employable and in high demand. I learnt Java from the Java core series many years ago. They offer two separate books: One is for Java fundamentals , the other is for advanced Java features. Focus on the fundamentals in this phase. Say you write a very simple program that just adds two integers and prints the result to the screen. In python, this would look like this: You understand that a computer running your code should output How does addition actually happen? At the end of the day, a computer is just a collection of hardware chips and wires. How can a computer really understand your code? The fact of the matter is, your code is just the tip of the iceberg. There are a lot of other layers under your code that together make the whole thing works the way you expect it to work. At this level, you need a solid understanding of all the layers of the stack starting from your code all the way down to the hardware layer. The book covers hardware, compilers, linkers, and operating systems at a very basic level which makes it very beginner friendly. It walks you through the steps of creating your first programming language, creating a compiler and a linker for it, and then creating an operating system. You understand how hardware is eventually going to run your code. You know that you have limited hardware resources and you understand the value of utilizing the available resources efficiently. Studying algorithms and data structures will teach you how to write code in a way that makes your code more efficient however you define efficiency: The skills you are going learn at this phase are one of the major differentiators that separates average coders from solid programmers. I personally like Introduction to Algorithms: A Creative Approach by Udi Manber. It is well written and includes a wealth of exercises. However, some people prefer to read books that are specific to their preferred language. Most useful code communicates with other computers either in a local network or the internet. Programmers need to have a very solid foundation of how computer networking works. I came across, in my opinion, the best networking book when I was a senior undergrad. It helped me overcome the bad teaching skills of my professor at the time and his dry text book. I still go back to this book every now and then if I need a refresher. If you are following this list in order, then by now you should have a very broad idea of the role of an operating system in the stack. But now is the time to have a deeper understanding of operating systems. You need some basic knowledge of C though because the majority of operating systems are written in C. This is a very dense topic and understanding all the details of all the aspects of operating systems is very time consuming. In this level, you will be learning new skills while you improve the skills you learnt as a programmer. Distributed systems is about building and architecting software systems that are scalable and that can tolerate failures at the same time. This requires you to think of the bigger picture rather than focusing on how to build the individual components programmers and coders can do that. For example, think about building a search engine service like Google for some text files that exist in your laptop only. This service that you built is going to be listening to search queries that it receives over the network, search your files for the query, and respond with the results. In fact, this is not a hard thing to do. Now imagine that more and more people become interested in your service and they start using it. Not only that but the size and number of files you are searching through begins to dramatically grow. What happens if your laptop that hosts the search service fails? Distributed systems is about creating an army of computers that work together to form a specific task in our example, the search service. It allows you to create scalable systems that can handle more requests or more data. At the same time, it provides redundancy that would be useful in case any one or more machine fail. By far, this link is the best resource I have found on the subject disclaimer: I have skimmed through it and it covers most of the important topics. With that said, Distributed Systems is a field where experience matters a lot.

Chapter 3 : Illinois Computer Science

Students can choose from among three computer science degrees: 1) the Bachelor of Arts in the College of Arts and Sciences, 2) the Bachelor of Science in the College of Arts and Sciences and 3) the Bachelor of Science in Computer Science and Engineering, which is offered by the College of Engineering.

Charles Babbage sometimes referred to as the "father of computing". Machines for calculating fixed numerical tasks such as the abacus have existed since antiquity, aiding in computations such as multiplication and division. Further, algorithms for performing computations have existed since antiquity, even before the development of sophisticated computing equipment. Wilhelm Schickard designed and constructed the first working mechanical calculator in In , Thomas de Colmar launched the mechanical calculator industry [note 1] when he released his simplified arithmometer , which was the first calculating machine strong enough and reliable enough to be used daily in an office environment. Charles Babbage started the design of the first automatic mechanical calculator, his Difference Engine , in , which eventually gave him the idea of the first programmable mechanical calculator, his Analytical Engine. Computer science began to be established as a distinct academic discipline in the s and early s. The first computer science degree program in the United States was formed at Purdue University in Although many initially believed it was impossible that computers themselves could actually be a scientific field of study, in the late fifties it gradually became accepted among the greater academic population. Initially, computers were quite costly, and some degree of humanitarian aid was needed for efficient useâ€”in part from professional computer operators. As computer adoption became more widespread and affordable, less human assistance was needed for common usage. Contributions[edit] The German military used the Enigma machine shown here during World War II for communications they wanted kept secret. The start of the " Digital Revolution ", which includes the current Information Age and the Internet. It also enabled advanced study of the mind, and mapping of the human genome became possible with the Human Genome Project. Algorithmic trading has increased the efficiency and liquidity of financial markets by using artificial intelligence , machine learning , and other statistical and numerical techniques on a large scale. Even films that feature no explicit CGI are usually "filmed" now on digital cameras , or edited or post-processed using a digital video editor. Modern computers enable optimization of such designs as complete aircraft. Notable in electrical and electronic circuit design are SPICE, as well as software for physical realization of new or modified designs. The latter includes essential design software for integrated circuits. There are many applications of AI, some of which can be seen at home, such as robotic vacuum cleaners. It is also present in video games and on the modern battlefield in drones, anti-missile systems, and squad support robots. Humanâ€”computer interaction combines novel algorithms with design strategies that enable rapid human performance, low error rates, ease in learning, and high satisfaction. Researchers use ethnographic observation and automated data collection to understand user needs, then conduct usability tests to refine designs. Key innovations include the direct manipulation , selectable web links, touchscreen designs, mobile applications, and virtual reality. Because of this, several alternative names have been proposed. Danish scientist Peter Naur suggested the term datalogy, [32] to reflect the fact that the scientific discipline revolves around data and data treatment, while not necessarily involving computers. The first scientific institution to use the term was the Department of Datalogy at the University of Copenhagen, founded in , with Peter Naur being the first professor in datalogy. The term is used mainly in the Scandinavian countries. An alternative term, also proposed by Naur, is data science ; this is now used for a distinct field of data analysis, including statistics and databases. Also, in the early days of computing, a number of terms for the practitioners of the field of computing were suggested in the Communications of the ACMâ€”turingineer, turologist, flow-charts-man, applied meta-mathematician, and applied epistemologist. For example, the study of computer hardware is usually considered part of computer engineering , while the study of commercial computer systems and their deployment is often called information technology or information systems. However, there has been much cross-fertilization of ideas between the various computer-related disciplines. Computer science research also often intersects other disciplines, such as philosophy, cognitive science , linguistics , mathematics , physics , biology , statistics , and logic. Computer science is considered by some to have a much closer relationship with mathematics than many scientific disciplines, with some observers saying that computing is a mathematical science. Computer science departments with a mathematics emphasis and with a numerical orientation consider alignment with computational science. Both types of departments tend to make efforts to bridge the field educationally if not across all research. Philosophy of computer science A number of computer scientists have argued for the distinction of three separate paradigms in computer science. Peter Wegner argued that those paradigms are science, technology, and mathematics. Eden described them as the "rationalist paradigm" which treats computer science as a branch of mathematics, which is prevalent in theoretical computer science, and mainly employs deductive reasoning , the "technocratic paradigm" which might be found in engineering approaches, most prominently in software engineering , and the "scientific paradigm" which approaches computer-related artifacts from the empirical perspective of natural sciences , identifiable in some branches of artificial intelligence. Outline of computer science As a discipline, computer science spans a range of topics from theoretical studies of algorithms and the limits of computation to the practical issues of implementing computing systems in hardware and software. In addition to these four areas, CSAB also identifies fields such as software engineering, artificial intelligence, computer networking and communication, database systems, parallel computation, distributed computation, humanâ€”computer interaction, computer graphics, operating systems, and numerical and symbolic computation as being important areas of computer science. Theoretical computer science Theoretical Computer Science is mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies. All studies related to mathematical, logic and formal concepts and methods could be considered as theoretical computer science, provided that the motivation is clearly drawn from the field of computing. Data structures and algorithms[edit] Data structures and algorithms are the study of commonly used computational methods and their computational efficiency.

Chapter 4 : CSE Computer Science I

The Master of Computer Science is a non-thesis degree that requires 32 credit hours of coursework. Students can complete the eight courses required for the Master of Computer Science at their own pace, in as little as one year or as many as five years.

Internet Technologies and their Social Impact. Examines current Internet technologies and social implications at the individual, group, and societal level. Blogs, wikis, sharing of video, photos, and music, e-commerce, social networking, gaming, and virtual environments. Issues include privacy, trust, identity, reputation, governance, copyright, and malicious behavior. Human Factors for the Web. Principles of human-computer interaction in evaluating, designing, and developing information presented on the World Wide Web. User characteristics, usability analysis, navigation and organization. Color, typography, multimedia, information visualization, prototyping, user studies, evaluation strategies. Global Disruption and Information Technology. Explores how new forms of information technology may support transition to a sustainable civilization. Activities involve reading, writing, discussion, and final project. Boolean Logic and Discrete Structures. Relations and their properties; Boolean algebras, formal languages; finite automata. High school mathematics through trigonometry. Computer Science Engineering Majors have first consideration for enrollment. Discrete Mathematics for Computer Science. Covers essential tools from discrete mathematics used in computer science with an emphasis on the process of abstracting computational problems and analyzing them mathematically. Topics include mathematical induction, combinatorics, and recurrence relations. Computer Engineering majors have second consideration. Matrices and linear transformations, systems of linear equations, determinants, linear vector spaces, eigenvalues and eigenvectors, orthogonal matrices, diagonalization, and least squares. Topics will be taught primarily from an algorithmic perspective, including computational solutions, applications, and numerical error analysis. Capabilities and limitations of information technology. Representing information in digital form. Overview of computer organization, Internet, operating systems, software. Human-computer interaction and social impact. The Internet and Public Policy. How the Internet works. Current public policy issues concerning the Internet. Interactions between information technology, economics, and law. Case studies about Internet and communications policy. Foundational principles of computer science for students with no computing background that are interested in a related career. Big ideas of computing explored, including programming through use of sequential, conditional, iterative logic. Good computational practices, problem solving, and organization discussed. Introduction to fundamental concepts and techniques for writing software in a high-level programming language. Programming with Software Libraries. Using library modules for applications such as graphics, sound, GUI, database, Web, and network programming. CSE 41 with a grade of C or better Restriction: Python Programming and Libraries Accelerated. Introduces Python syntax and semantics for fundamental programming concepts. Constructing programs for varied problems and environments. Accelerated course for students with previous programming background. AP Computer Science A. AP Computer Science A with a minimum score of 3. Placement via a transfer course in computer programming or equivalent experience may also be accepted upon review. Intermediate-level language features and programming concepts for larger, more complex, higher-quality software. Functional programming, name spaces, modules, class protocols, inheritance, iterators, generators, operator overloading, reflection. Analysis of time and space efficiency. CSE 42 with a grade of C or better Restriction: Emphasis on object-oriented programming, using standard libraries, and programming with manual garbage collection. CSE 43 with a grade of C or better. Programming in Java as a Second Language. An introduction to the lexical, syntactic, semantic, and pragmatic characteristics of the Java language for experienced programmers. Emphasis on object-oriented programming, using standard libraries, and programming with automatic garbage collection. Data Structure Implementation and Analysis. Focuses on implementation and mathematical analysis of fundamental data structures and algorithms. Covers storage allocation and memory management techniques. CSE 45C with a grade of C or better. Multilevel view of system hardware and software. Operation and interconnection of hardware elements. Instruction sets and addressing modes. Virtual memory and operating systems. Laboratory work using low-level programming languages. Principles in System Design. Principles and practice of engineering of computer software and hardware systems. Topics include techniques for controlling complexity; strong modularity using client-server design, virtual memory, and threads; networks; coordination of parallel activities; security and encryption; and performance optimizations.

Chapter 5 : What is Computer Science? | Study Computer Science in the US

The computer science major equips students to master common computer languages used to create programs and to understand the logic and structure of languages so that they can more easily learn new computer languages.

Introduction We will begin this course by identifying our motivation for learning fundamental programming concepts and learning the history of programming languages in general. We will then discuss the hardware the physical devices that make up the computer and software Operating Systems and applications that run on the computer of a computer. We will conclude with a brief discussion of the Java programming language, which we will use throughout the rest of this course. By the end of this unit, you will have a strong understanding of the history of programming and be well prepared to learn about programming concepts in greater detail. Completing this unit should take you approximately 16 hours. Since Object-Oriented programming OO is currently one of the most popular programming paradigms, you will need to learn its fundamental concepts in order to build a career in Computer Science. This unit will begin with a discussion of what makes OO programming so unique, and why its advantages have made it the industry-standard paradigm for newly designed programs. We will then discuss the fundamental concepts of OO and relate them back to Java. By the end of this unit, you will have a strong understanding of what Object-Oriented programming is, how it relates to Java, and why it is employed. Completing this unit should take you approximately 10 hours. Java Fundamentals Now that you have a basic understanding of OO programming, we will move on to the fundamental concepts of the programming language we will be studying this semester: The Java-related concepts you will learn in this unit are in many cases directly transferable to a number of other languages. In addition, we will also learn about two different styles of adding comments to the code. By the end of this unit, you should have a fundamental understanding of Java basics and be prepared to utilize those concepts later in the course. Completing this unit should take you approximately 22 hours. Relational and Logical Operators in Java In this unit, we will discuss relational and logical operators in Java, which provide the foundation for topics like control structures which we will further discuss in Unit 5. In this unit, we will start by taking a look at operator notation. We will then discuss relational operators as they apply to both numeric operands and object operands before concluding the unit with an introduction to logical operators. By the end of this unit, you should be able to perform comparisons and logic functions in Java and have a fundamental understanding of how they are employed. Completing this unit should take you approximately 9 hours. Control Structures Control structures dictate what the behavior of a program will be under what circumstances. Control structures belong to one of two families: Control structures like if-then-else and switch the program to behave differently based on the data that they are fed. The while and for loops allow you to repeat a block of code as often as it is needed. As you can see, that functionality can be very useful when designing complex programs. We will also discuss some advanced topics, such as nesting and scope. By the end of this unit, you should be able to draw from the information you learned in the previous unit to create a control structure, which will allow you to create more complex and useful programs. Completing this unit should take you approximately 20 hours. User-Defined Methods In addition to the methods predefined in Java, we can write user-defined methods. In this unit, we will discuss how to name a method, declare a parameter list, and specify the return type. This unit introduces the scope of variables as well. By the end of this unit, you will have a strong understanding of how to define and call a method. Completing this unit should take you approximately 11 hours. Arrays This unit discusses Arrays. An Array is a fixed-size data structure that allows elements of same data type to be stored in it. Each array element has a unique index associated with the value it stores. Arrays are commonly used in a loop structure such as for loops. In addition, this unit introduces two-dimensional arrays and its applications. Completing this unit should take you approximately 7 hours. Input and output techniques allow programmers to design more complex and useful programs. We will then identify the common pitfalls and design concepts that you should keep in mind as a programmer. By the end of this unit, you will have a strong understanding of how to write and read from a file and how to write a Java program that performs these functions. Exception handling mechanism allows a program to continue executing, even if an error occurs in the program, instead of terminating it abruptly.

Chapter 6 : Highest-Paying Jobs For Computer Science Majors in | PayScale

Archived Electrical Engineering and Computer Science Courses Some prior versions of courses listed above have been archived in OCW's DSpace@MIT repository for long-term access and preservation. Links to archived prior versions of a course may be found on that course's "Other Versions" tab.

Chapter 7 : CSC - Computer Science I: (C++) - Colorado Community Colleges Online

Illinois Computer Science's eighth annual Alumni Awards Ceremony and Banquet recognized 13 alumni and faculty who have made professional, technical, educational, or service contributions that bring distinction to themselves, the department, and the university.

Chapter 8 : Electrical Engineering and Computer Science | MIT OpenCourseWare | Free Online Course Ma

Computer Science faculty and students working in partnership with each other and with interdisciplinary colleagues will help address significant local, regional, and national problems through the use of this flexible, robust discipline.

Chapter 9 : Difference Between IT and Computer science | Difference Between

More than 65% of students will work in jobs that don't even exist today.Â¹ We want to help prepare them for that future by getting them excited about where computer science (CS) can take them.