Recommendations of the CBSE Computer Science Curriculum Review Committee

Prof. Smruti R. Sarangi, Computer Science and Engineering, IIT Delhi
Prof. Vikram Goyal, Computer Science and Engineering, IIIT Delhi

January 8, 2018

Contents

1 Scope of this Document

This document contains the draft recommendations of a curriculum review committee for the computer science courses that are a part of the current CBSE curriculum. It is meant for a general readership. The committee has tried its best to explain technical terms in common language without sacrificing rigour. Note that this document represents the views of the curriculum review committee. CBSE, or any other organization do not bear any responsibility for any errors of omission or commission in this document. This report has been prepared by an independent committee, and CBSE does not necessarily endorse all the views of the authors of this document. This is an academic document, and does not intend to tread the area of administrative issues, and should be treated as a recommendation, rather than a policy document.

Novel features of the recommendations:

1. In classes 9 and 10, have one course called *Computer Applications*.
   (a) Introduce basic programming logic using an educational language called Scratch or Python (for special needs children).
2. Modify the Computer Science (CS) course (in classes 11 and 12) as follows:
   (a) Teach Python instead of C++/Java.
   (b) Change the focus from learning a language to learning logic and problem solving skills.
3. Upgrade the Informatics Practices (IP) course (in classes 11 and 12)
   (a) Replace Java with Python
   (b) Add a significant *Data Science* component that has explicit focus on data analysis, statistics, and visualization. Decrease the focus on computer hardware, networking and other low level details.
   (c) Introduce a small module on software engineering and business processes in the IP curriculum.
4. Discontinue the course on Multimedia and Web Technology (in classes 11 and 12).
5. Discontinue the older curriculum and introduce the new curriculum for class 9 in 2018. Introduce the new curriculum for class 11 in 2018, and discontinue the older curriculum (for class 11) in 2019. This means that in 2018, students entering class 11 will have a choice between the new and old curricula (only CS and IP).
6. For all classes teach concepts in cyber safety, cyber etiquette, and responsible use of social media.

We would like to acknowledge CBSE for connecting us with schools, and school teachers for getting their valuable inputs.

2 Introduction

2.1 Background

CBSE (Central Board of Secondary Education) is a large nation wide school board in India, which is operated by the Central (Union) Government of India. As of June 2017, 14860 private schools, 4456 public (government/aided) schools, and 211 schools in 25 foreign countries were affiliated with the board. The term *affiliation* means that school managements agree to operate their institutions as per CBSE's guidelines and by laws, follow CBSE's prescribed curriculum, and facilitate their students to write nationwide board exams in classes 10 and 12. India has other nation wide boards such as CISCE, and NIOS (for open schooling); however, CBSE is the largest. In addition, a lot of states in India have their own state boards, which function in a similar manner. To summarize, CBSE's role is multifaceted and is composed of regulation, standardization, making curricular recommendations, and lastly conducting large nationwide board exams.

More than a million students appeared in the board exam for students in class 12 (conducted this year), where they had a choice of 159 subjects.

Specifically, the CBSE curriculum has three subjects with regards to computer science and computer applications from classes 9 to 12 (three subjects/courses per class). Students typically choose one of the courses if it is offered in their school. CBSE has had courses in the area of computer science from the late nineties, and these courses have been highly subscribed to in the schools that they were being offered.

Please note that offering computer science subjects in school is a global trend, and many advanced countries offer such subjects at much younger ages – even primary school. In line with the global trend, CBSE has a strong focus in these areas, and wishes to have a very fruitful and effective curriculum. Similar initiatives are also being followed in other countries. Academics in the United States have recently in 2016 put together a curricular framework [Committee, 2016]. There are similar efforts in the UK [Brown et al., 2014, Jones, 2013], Israel [Ezer and Harel, 1999], Germany [Hubwieser, 2012], New Zealand [Bell et al., 2010], Australia [Australian Curriculum Assessment and Reporting Authority, 2017], and all over Europe [Heintz et al., 2016]. The apex body of computer scientists, ACM (Association of Computing Machinery), had suggested a model curriculum [Tucker, 2003] for schools way back in 2003. Many of the curricular interventions today are heavily inspired by ACM's model curriculum.

Therefore a need was felt to modernize the current curriculum, bring it in line with international curricula, and furthermore correct some of the issues in the current curricula. A meeting was conducted on September 18^{th}, 2017, in Shastri Bhavan, New Delhi (office of the Ministry of Human Resources) under the chairmanship of the current CBSE chairperson, Ms. Anita Karwal. The other attendees included CBSE office bearers, representatives from industry, and academia.

The CBSE chairperson laid out a vision for a new curriculum and solicited the feedback of the attendees. There was a broad agreement with regards to the need for having a new curriculum such that CBSE students can get the best in terms of computer science (henceforth abbreviated as CS) education. It was decided that the committee will be co-chaired by Prof. Smruti R. Sarangi (Computer Science and Engineering, IIT Delhi) and Prof. Vikram Goyal (Associate Dean Student Affairs, IIIT Delhi). The role of overall mentorship was given to Prof. Pankaj Jalote (current director of IIIT Delhi, ex-Professor of IIT Delhi and IIT Kanpur).

The following was the vision that was presented in the meeting.

2.2 Vision

The honourable CBSE chairperson, Ms. Anita Karwal, and other CBSE officials were cognizant of the fact that CS is a very fast moving field. The sheer amount of change that the area has witnessed is totally unprecedented. Similar changes in physics and chemistry were happening at the turn of the 20^{th} century; however, the school curricula for these traditional subjects have now more or less stabilized. Given the massive rate of change, there is a need (and probably will continue to be) to revamp the curriculum.

During the meeting the CBSE officials were also apprised of the fact that the learning outcomes in school level CS education are not the best, and a significant scope of improvement exists.

Keeping all of these issues in mind, the chairperson outlined her vision.

- Create a futuristic curriculum that is in line with technological trends and international standards.
- Ensure great learning outcomes.
- Take a look at the existing curriculum and remove any redundant parts. Streamline the curriculum to ensure more efficient delivery.

2.3 Composition of the Curriculum Review Committee

The chairpersons of the committee were given the task to constitute a committee. They took the help of CBSE office bearers for getting a list of ex-officio members and representatives from schools. Subsequently, they started a process of contacting eminent experts in both academia and industry. Based on constraints

such as willingness, expertise, and availability, experts from academia and industry were added to the committee. The details of the committee are shown in Table 1.

Special emphasis was laid on having a committee that was as diverse as possible. The chairpersons tried to construct a committee that is not particularly biased towards any region, or people who subscribe to a certain school of thought. For industry members, a diverse set of industries were chosen: research labs, product development labs, startups, and companies that offer financial services.

Making the curriculum friendly towards children with special needs is one of the explicit goals of this committee. It was a pleasure to include Dr. Ramana Polavarapu who is an eminent computer scientist and is visually challenged, and Ms. Sapna Sukul who is the founder of a reputed special school in Delhi – Sparsh. Wherever, additional help was required, international experts were contacted and their views were taken into account (see Table 2).

2.4 Organisation of this Document

In Section 3 we shall discuss our research and data collection methodology, then we shall move on to Section 4 and Section 5, where we discuss the need for a new curriculum and the main guiding principles for designing a new curriculum respectively. Subsequently, we present the recommendations for the new curriculum in Section 6, and finally discuss methods to operationalise the curriculum in Section 7.

3 Research and Data Collection Methodology

For such a large exercise, it is necessary to make use of all credible sources of information. We first looked at sources of written information.

3.1 Research Papers, White Papers, and Media Reports

Existing curricular documents and textbooks The committee went through all the curricular documents for the current curriculum, and also went through most of the popular textbooks. In addition, the committee looked at most of the recent exam papers, and test preparation materials. A thorough study of the exam papers and marking standards gave the committee an idea of the difficulty level in the current set of courses.

Study of the ACM model curriculum and curricula in foreign countries ACM (Association for Computing Machinery) is the apex body of computer scientists in the world. It had setup a task force for drafting a K12 (kindergarten to class 12) curriculum. The committee read through the final recommendations [Committee, 2016, Tucker, 2003], and also considered the curriculum followed in 15 advanced countries [Heintz et al., 2016, Hubwieser et al., 2015b, Hubwieser et al., 2014, Jones et al., 2011].

Study of research papers in the area of K12 CS education The committee members were particularly influenced by the following research papers.

1. Paper on the current status of CS education in India, and its associated issues and challenges [Raman et al., 2015].
2. The committee read through a lot of the work by Prof. Peter Hubwieser (TU Munich), and was heavily influenced by his in-depth analysis and coverage of topics and relevant issues. Some of his important papers are [Hubwieser et al., 2014, Hubwieser et al., 2016, Hubwieser et al., 2015a, Hubwieser et al., 2015b]
3. The committee was also heavily influenced with the work of Dr. Simon Peyton Jones from Microsoft Research, Cambridge [Jones et al., 2011, Jones, 2013].

S. No.	Name	Qualification	Designation
Mentor			
1.	Prof. Pankaj Jalote	Ph.D (UIUC, USA)	Director, IIIT Delhi (ex-Professor IIT Delhi and Kanpur), INAE and IEEE Fellow
Co-Chairpersons			
2.	Prof. Smruti R. Sarangi	Ph.D (UIUC, USA)	Associate Professor, jointly in Computer Science and Electrical Engg., IIT Delhi
3.	Prof. Vikram Goyal	Ph.D (IIT Delhi)	Associate Dean Student Affairs and Associate Professor, IIIT Delhi
Members from Academia and Industry			
4.	Prof. Madhavan Mukund	Ph.D (Aarhus Univ., Denmark)	Professor and Dean of Studies, Chennai Mathematical Institute
5.	Prof. Kishore Kothapalli	Ph.D (John Hopkins University, USA)	Associate Professor, IIIT, Hyderabad
6.	Prof. Amit Kumar	Ph.D (Cornell Univ., USA)	Professor, Computer Science Engg., IIT Delhi
7.	Prof. Yogesh Simmhan	Ph.D (Indiana Univ., USA)	Assistant Professor, Computational and Data Sciences, IISc, Bangalore
8.	Prof. Chetan Arora	Ph.D (IIT Delhi)	Assistant Professor, IIT Delhi
9.	Prof. Sushmita Ruj	Ph.D (ISI, Kolkata)	Assistant Professor, ISI, Kolkata
10.	Dr. Partha Dutta	Ph.D (EPFL, Switzerland)	Vice President, Swiss Re-insurance (5 years exp. in IBM Research Labs)
11.	Dr. Ramana Polavarapu	Ph.D (UC Davis, USA)	Vice President, Goldman Sachs (5 years exp. in IBM Research Labs)
12.	Dr. Subrat Panda	Ph.D (IIT Kharagpur)	Principal Architect, Capillary Tech., (3 years exp. in IBM, 2 years exp. in NVidia)
13.	Dr. Sameep Mehta	Ph.D (Ohio State Univ., USA)	Senior Researcher, IBM Research Labs, Delhi
Members from Schools			
1.	Mr. Mukesh Kumar	DPS R. K. Puram, Delhi	
2.	Ms. Purvi Srivastava	Ganga International School, Delhi	
3.	Ms. Divya Jain	Apeejay School, Noida	
4.	Ms. Chetna Khanna	Shadley Public School, Delhi (till March 2017)	
5.	Ms. Purnima	Kendriya Vidyalaya, Paschchim Vihar, Delhi	
6.	Mr. R. K. Tiwari	Vivekananda School, Delhi	
7.	Mr. Shailender Gupta	Lt. Col Mehar Little Angels Sr. Sec. School, Delhi	
Members from Disability Organizations			
1.	Ms. Sapna Sukul	Sparsh Special Needs School, Delhi	
Ex-Officio Members			
1.	Prof. Om Vikas	Chairman NAAC, ex-Director of IIIT Gwalior	
2.	Dr. Biswajeet Saha	Additional Director, CBSE	
3.	Mr. Subhash Chand Garg	Deputy Director, CBSE	

Table 1: Constitution of the curriculum review committee

Study of media reports – conventional media and social media The members went through reports

S. No.	Name	Designation	Nature of recommendation
1.	Prof. Pawan Sinha	Professor, Department of Brain and Cognitive Sciences, MIT	Curricular modifications for children with cognitive abnormalities and Autism.
2.	Prof. Uday Khedkar	Professor and HoD, Computer Science Deptt., IIT Mumbai	Basic principles of designing a curriculum for schools
3.	Prof. Anupam Basu	Professor, Computer Science and Engg., IIT Kharagpur	Principles of designing a curriculum and methods to make it friendly towards children with Cerebral Palsy
4.	Prof. D. R. Prabhu	Professor, Sathya Sai Center for Human Excellence, Chickballapur	Recommendation for making the curriculum friendly for children with Autism
5.	Dr. Raghu Cavale	Vice President and India Business Head, Infosys	Insights into developing a curriculum that will stand the test of time.
6.	Prof. Neelima Gupta	Professor, Computer Science, Delhi University	Her experiences as the chairperson of the previous curriculum review committee and ideas on how to develop the curriculum for the future.
7.	Prof. Tapan Gandhi	Assistant Professor, Electrical Engineering, IIT Delhi	Curricular modifications for teaching computer science to children with Autism
8.	Prof. Maya Ramanath	Assistant Professor, Computer Science, IIT Delhi	Help in designing the data management curriculum
9.	Prof. Aaditeshwar Seth	Associate Professor, Computer Science, IIT Delhi	Help in designing the computer networks curriculum
10.	Prof. Vibha Arora	Associate Professor, Humanities and Social Sciences, IIT Delhi	Help in designing the curriculum on society, law, and cyber-etiquette
11.	Shubhankar Suman Singh (IIT Delhi), Omais Shafi Pandith (IIT Delhi) and Siddharth Dawar (IIIT Delhi)	Ph.D students	They studied international curricula, assessed the merits of different software platforms, and provided vital inputs throughout the process.

Table 2: List of external experts

published in the popular press and social media regarding the public perception of the current curriculum. It is true that as compared to research papers and government documents, reports – particularly in social media – can sometimes have questionable credibility. Nevertheless, the members still perused through these articles with an open mind, and always tried to correlate the reports with more credible sources in an unbiased manner.

3.2 School Visits

After poring through all the written information that the members could find, subsets of the committee proceeded to having person-to-person interactions with diferent stakeholders. It was decided that the members will visit schools and talk to principals, teachers, and students. The choice of schools was an issue. It was decided that the committee shall divide schools into three bands broadly based on the performance of the students in the Class 12 board exam (computer science subjects). Let us refer to them as level 1 (highest

marks), level 2, and level 3(lowest marks). The top 20 students in level 1 schools typically are expected to score more than 95% based on historical trends, a similar cohort in level 2 schools is expected to score between 80 to 95%, and the cohort in level 3 schools is expected to score somewhere between 65% and 80%. Considerations of geographical diversity was also an issue. In principle, it is possible that schools in Delhi might have a very different opinion as compared to schools in Bangalore.

In response, three teams were created – one in Delhi, one in Hyderabad, and one in Bangalore. Each team was tasked to visit at least one school in each of the levels. The committee did not observe any significant geographical diversity. The observations were pretty much the same across geographies. They were only dependent on the level of the school.

The same protocol was followed for all the visits.

- First talk to the principal of the school for 10-15 minutes.
- Interact with the teachers for roughly 15-20 minutes.
- Have a closed door interaction with roughly 10-15 students from classes 11 and 12.
- During the student interaction the committee members collected their general views about the curriculum. Then they proceeded to ask them some general computer science questions.
- The idea was to ask them a fairly difficult question, and see if they can get to the answer. None of them were expected to know the answer. However, the exercise was to see if they have enough skills to navigate to the answer with hints from the committee members. This is a standard method of interacting with students particularly when the aim is to see how easily a person can be trained to pick up advanced skills. Along the way the aim of the committee members was to passively assess the students without causing them any discomfort or giving them a feeling that they were being evaluated.
- The students had a great time, and the interaction proceeded in a very jovial and amicable fashion. The committee members were able to assess the current level of the students, and their potential. In specific the students were asked the following computer science questions: merge two sorted arrays, and use this algorithm to sort an unsorted array (basically move towards mergesort). In layman's terms, here is an alternative explanation. In this case, an *array* is an ordered list of numbers. We consider an array where the numbers are not sorted. Then we ask the students to suggest a method that can be used to sort the array. Since the computer is a very dumb machine it can only do basic things like comparing two numbers, or moving one number from one location to the other. Using such simple operations the task was to sort the array. The members saw mixed results (discussed in Section 4.3).
- In some schools the members were able to talk to special educators, and get an idea of the issues faced by children with special needs.

The committee members visited a total of 7 schools spanning from high end private schools to government schools. In each school they met a group of at least 10 students, the teachers, and the principal. Once they observed that the nature of interaction was getting very predictable, they concluded that the point of diminishing returns has arrived, and visiting more schools was not necessary.

3.3 Interaction with Parents

The last step was a structured interaction with parents. This interaction was facilitated by the schools. The members talked to 10-15 parents across social strata. Here also, once the nature of the interaction started to get very predictable, and no new information was coming in, the process was stopped.

The parents were extremely co-operative and supportive. They welcomed the idea of a curriculum review and wanted a very modern curriculum that ensured good learning outcomes. They were furthermore very happy with the consultative approach that was being followed. They however had several words of caution for the committee. They indicated that as of today, parents are in general very aware. They often visit the

CBSE website, and share information about the curriculum with other parents via Facebook and Whatsapp. As a result they will be quick to react to any changes in the new curriculum.

Their first point of contact is the school teacher. If the school teacher is not aware of the proposed changes, or of the rationale behind the changes, then he or she might not convey the correct impression to the parents. This will spread fear, mistrust, and confusion. This is to be avoided. In the parents view nothing should come as a surprise. Everything should be introduced after a prior notice. Any surprise move by the board will encounter a certain amount of resistance.

They were of the view that this document should be made public such that parents at least have an alternative source of information. If this document is easily readable then most parents and their wards can understand the rationale behind designing a revised curriculum. This will help clear any doubts and misconceptions. They further advocated a process where public feedback is collected, and important changes are incorporated into the curriculum. These processes in their view would create a fair amount of public trust, and ensure wider public participation.

Finally, some parents were of the view that pilot courses should be run in various schools across the country before a curriculum is fully ratified. Given logistical issues, and the legal sanctity of the final degree, the idea seemed hard to realize.

To summarize, the parents wanted a clear, transparent, consultative, and participatory process where all stakeholders are taken into account.

3.4 Unstructured Interaction

Along with structured interactions, the members had many unstructured interactions where they asked their friends, neighbours, relatives, and colleagues about their perceptions of the CBSE CS curriculum. In addition, unstructured discussions were held with children of colleagues who are currently studying CS in CBSE schools, and their friends. In addition, there were 7 school teachers in the committee who gave a lot of inputs, and were more than available to share their experiences.

One important question that arises in such an exercise is when to stop. There is no hard and fast rule per se. However, when we stop getting new information, and all the sources of information have been exhausted, it is possible to conclude that most of the knowledge with regards to the problem at hand has been gathered. A similar approach was followed here. For each of the areas, the committee stopped when there was a perception that additional efforts and interactions are not expected to yield any new information.

4 Need for a Revised Curriculum

Before talking about the need for a new curriculum, we should have a clear idea of the shortcomings of the current curriculum from a pedagogical (related to teaching) and logistical point of view. Furthermore, to understand the pros and cons of a curriculum, we need to compare it with a known benchmark. Hence, let us start out by looking at international curricula and trends in computer science teaching.

4.1 Trends in CS Teaching in Schools (Worldwide)

Unlike universities, teaching computer science in schools is not considered a core requirement. In many countries including advanced countries, prior to 2010, computer science was not considered to be a core subject. Some countries had compulsory courses in primary school, some countries made computer science compulsory in high school, some did both, and some did none. Furthermore, there was also no clear consensus regarding what should be taught. Most of the apex bodies such as IEEE and ACM were focused on college education, and did not have well defined documents for K12 education. In comparison, CBSE has had computer science courses for at least the last 25 years. These courses have steadily been redefined and modernized. Similar exercises are being carried out in other countries as well.

Specifically, the years from 2012 to 2016 have been watershed years for CS teaching in schools. In this period major standards bodies have defined curricula for schools, countries have undertaken major curriculum revision exercises, and leaders in the technology industry have openly spoken out in support of introducing CS education in schools. As computers become more ubiquitous, introducing some CS education in schools is becoming very important. Also, there has been a shift from traditional approaches that focus on computer literacy, and computer applications to teaching some of the formal concepts that underlie computer science. Shifting from a curriculum that was semi-formal in nature to a formal curriculum has its merits. This is because people from almost all walks of life will be doing something with computers at their home and work. It is thus necessary to teach them classic timeless skills such that they can use computers more meaningfully. To summarize, there is a shift from teaching ICT (Information and Communication Technologies), which is primarily tools and applications, to more formal concepts.

To avoid confusion, let us explain some of the terms that we shall use in our subsequent discussion. Note that these terms have been used in different contexts in the literature. However, let us standardize their meanings for the sake of better understanding in this document.

Term	Full Form	Explanation
ICT	Information and communication technologies	Study of computer applications and methods to communicate data.
FCS	Formal computer science	Mathematical aspects of computer science that deals mainly with problem solving.
CS	Computer science	An umbrella term used to describe all the courses that are related to understanding and using computers.

Please note that *FCS* is not a standard term. However, given the fact that there is a lot of ambiguity in the way that these terms are used, there was a need to define new terms. Let us differentiate between ICT and FCS with examples.

Topics in ICT Web designing, designing multimedia applications, description of computer systems

Topics in FCS Problem solving: find the largest number in a set of numbers, test if a number is prime, draw a circle on a screen; and abstract thinking: solve a problem by breaking it into smaller parts.

Most countries typically teach ICT topics such as specific computer applications – drawing, painting, word processing, games – at a younger level (primary school), and move to teaching more formal concepts

(FCS) at the high school level. This process was however not very standardized. Even though the apex body of computer scientists (ACM) had released a model curriculum in 2003, countries were slow to make formal changes in their curricula, particularly in high school. They still had curricula that was a mix of ICT and FCS, where the latter was under-emphasized. However, now there is a lot of traction in this area.

The shift from ICT to FCS was done [Heintz et al., 2016] by Australia in 2015, England in 2014, Norway in 2016 (pilot project), South Korea in 2007, Sweden in 2016, Poland in 2012 (as an elective), Germany in 2004 [Hubwieser, 2012], France in 2012 [Baron et al., 2014], Russia in the 90s [Khenner and Semakin, 2014], and USA in 2016 (released the ACM curriculum framework [Committee, 2016]). Note that in the US, every state has the freedom to decide its own curriculum. However, most states loosely follow ACM's recommendations, which suggest teaching FCS concepts in the higher grades. Note that Israel and Russia have advanced modules where extremely advanced (Olympiad level) concepts are taught.

This move to teaching formal mathematical concepts of CS in schools mainly happened for the following reasons [Bocconi et al., 2016, Wing, 2014]:

1. Computing is being regarded as a basic skill [Hubwieser et al., 2015a, Gardiner, 2014] at par with other mathematical skills. This is because computers have pervaded almost all areas of human life, and proficiency with computers is being considered critical to later life.

2. An approach purely based on ICT is too dependent on the vagaries of computer technology. ICT skills might not last the test of time, and have the risk of becoming obsolete once a new software comes to the market.

3. The traditional assumption was that students will learn FCS concepts in college, and schools are for acquiring basic skills in word processing and creating presentations. This assumption is gradually breaking down because almost all disciplines are increasingly using computers, and learning computer skills from scratch in college takes time away from other courses.

4. The introduction of *computational thinking* (CT) [Wing, 2014], which is defined as a method of thinking that is centred around the mathematical aspects of formal computer science benefits other areas also. Students can use the same reasoning in their mathematics courses, and often become better problem solvers in real life.

Given that technological trends are temporary, and concepts are timeless much of today's computer science education in schools is centred around the basic concepts in formal computer science (FCS): algorithms, language, machine, and information (see [Baron et al., 2014] and its cited references).

Let us explain these concepts that underlie all modern K12 curricula [Hubwieser et al., 2015b]

4.1.1 Algorithm, Language, Machine, and Information (from the French curriculum)

Algorithm The notion of *algorithms* is central to computer science; hence, it comes first. An algorithm is a procedure that needs to be followed by a computing device to process inputs, and achieve a desired output. Note that a computing device like a laptop or a desktop or a large server is a very dumb machine. It is only good for performing basic arithmetic operations, executing the same actions over and over again (*iteration*), and making simple decisions (*conditionals*). However, the greatness is that it can do such dumb things billions of times a second. This is where it outsmarts our brilliant brains. Directing such a dumb device to perform intelligent tasks is achieved by creating an algorithm.

Language Unlike human beings computers do not understand natural languages. They however understand programs written in computer languages such as C++, Java, and Python. We write programs to implement algorithms. Such programs are written in a programming language, and it is thus essential to learn a programming language such that algorithmic concepts can be expressed and conveyed to a computer. Programming languages have two aspects: syntax (symbols used to express concepts, and rules used to write the language), and semantics (meaning associated with programming language terms). It is true that syntax is important (such as a '.' after every sentence in English); however,

semantics is even more important where we understand the meaning of different features of a given programming language. Semantics per se is a formal concept, has deep mathematical roots, and such concepts are typically common across programming languages. For example, a mango in English is an "aam" in Hindi (it is the same thing). **It is important to understand the mathematical meaning of concepts rather than just focus on the syntax**.

Machine This part deals with the way a computer system is designed. This covers networks (techniques to interconnect computers such as the internet), and computer architecture (design of computer hardware). Both of them are advanced topics. However, a very basic introduction can be provided at the school level.

Information This part deals with methods to represent and store information. Traditional databases, internet based information systems such as Google, and methods to analyse and interpret data are a part of this area.

The theme that is common to all the four areas – algorithm, language, machine, information – is a term called *computational thinking* [Bocconi et al., 2016], which is being increasingly used by curriculum committees all over the world. Quoting Prof. Jeannette Wing (Carnegie Mellon University) CT is defined as:

Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.

It is independent of the programming language, and independent of any particular hardware or software. **Computational thinking is a skill like bicycling and swimming, which once learnt is hard to forget, yet is hard to learn unless it is taught well.**

4.1.2 Structure of International Curricula

The ACM model curriculum advocates modifying existing curricula in mathematics, science, and social studies to incorporate some degree of critical algorithmic thinking till class 8 (Level I). Subsequently, in classes 9 and 10 (Level II), it recommends using various computer software to prepare documents, spreadsheets, presentations and design web pages. Students at this level are also supposed to have a basic understanding of computer hardware, networks, and cyber ethics. Classes 11 and 12 (Level III and IV) are almost exclusively meant for learning complex algorithmic and programming concepts. Most curricula broadly follow these guidelines including CBSE's existing curricula. The subsequent ACM curriculum framework document [Committee, 2016] expands on these areas and gives a road map from class 2 onwards. One advantage of this approach is that the learning curve is not very steep.

Advanced countries [Brown et al., 2014, Jones, 2013, Hubwieser et al., 2015b] typically start their CS education by the end of primary school, and thus have a lot of time to introduce concepts gradually. They start from typing and games, by class 9 they move to simple algorithmic concepts, and then introduce educational languages such as Alice and Scratch. These are very easy to learn and are based on a visual interface where simple concepts of iteration (doing a similar computation over and over again) can be learnt. Almost everywhere high school is dedicated to core FCS concepts, where algorithms and the details of computer hardware/networks are taught in detail.

There are some exceptions. Finland has an explicit focus on computer programming at the primary level [Hubwieser et al., 2015b]. Israel and Russia have extremely advanced courses at the school primarily for gifted students who are interested in appearing for different programming Olympiads.

4.2 History of CBSE's CS Curriculum

Let us start our clock at the turn of this century (1/1/2000). CBSE at that time offered three courses in each year from classes 9 to 12.

4.2.1 Classes 9 and 10

In classes 9 and 10, the three courses were as follows.

Course	Description
Foundation of Information Technology (FIT)	word processing, spread sheets, databases, website design using HTML, IT and society
Information and Communication Technology (ICT)	basics of the internet, GIMP image editing, website design using HTML, dynamic HTML and style sheets, network security
E-publishing and e-office	typing, word processing, spreadsheets, image editing with GIMP, E-publishing

These courses were designed as per the ICT paradigm. They were meant to give an introduction to computers to students who did not have prior exposure. Most of the key computer applications – word processors, spreadsheets, presentation software, web authoring, and image editing – were covered. Along with that some emphasis was laid on explaining some of the fundamentals of the internet and computing devices. Note that there was a strong overlap between these three courses, and they did not have a programming component. The committee made a note of this.

4.2.2 Classes 11 and 12

CBSE has three more courses for classes 11 and 12.

Course	Description
Computer Science(CS)	programming in C++, databases, basics of computer architecture and networking
Informatics Practices (IP)	basic Java programming, databases, basics of computer architecture and networking, IT applications
Multimedia and Web Technologies (MWT)	web development with HTML and PHP, web scripting languages: Javascript, databases, basics of computer architecture and networking, multimedia tools

These three courses were designed with three different kinds of students in mind. The students who opted for CS are expected to join courses in the Sciences and Engineering. Informatics Practices (IP) primarily prepares a student for the IT industry; however, it has a lesser amount of formal CS (FCS) content. Multimedia and web technologies (MWT) were meant for students who wanted an early introduction to web site design and working with images and videos.

4.3 Learning Outcomes and Subjective Experiences

Based on expectations set by international curricula, research papers, and the personal experiences of committee members while teaching in foreign countries, they proceeded to evaluate the learning outcomes in CBSE schools.

Recall that the committee had divided schools into three categories based on the marks that students have historically obtained in their 12^{th} class board exams (refer to Section 3.2). Level 1 schools had the highest marks and level 3 schools had the lowest.

4.3.1 Nature of the Student Interaction

In all schools, the students were asked the following questions:

1. Design an algorithm to merge two sorted arrays. An *array* is an ordered set of numbers. For example, we can define an array A to contain 5 elements(numbers) : $A = (10, 7, 6, 8, 19)$. As we see, array A contains 5 numbers. The question says that if we have two arrays that are already sorted, create an algorithm to merge the arrays and create a larger array, which is also sorted. Example: $A = (1, 3, 5)$, $B = (2, 4, 6)$. We can merge A and B to create a larger array, $C = (1, 2, 3, 4, 5, 6)$. Note that the elements in C are sorted.

   (a) How many loops are required?

   (b) Count the number of comparisons.

2. Design an algorithm to sort an unsorted array (*merge sort algorithm*) using the solution of the previous question. They are not expected to know the answer to this question. However, the idea was to see how far students can go with hints.

   (a) Recognize that a larger problem can be solved by breaking into two similar smaller problems. Split the problem into two smaller sub-problems (this is the basic idea of the *divide-and-conquer* technique).

   (b) Sort the smaller arrays.

   (c) Merge the sorted arrays.

   (d) Write the algorithm.

   (e) Count the number of steps.

3. Use similar divide-and-conquer techniques to compute x^n, which can be defined as $(\underbrace{x \times x \ldots \times x}_{n})$.

   Later if time permitted, the committee looked at similar problems.

Note that the for a student in class 11 and 12, these are fairly difficult problems. They are not expected to do them completely and independently. However, the aim of the exercise was two-fold. The first was to evaluate the *learning potential* of the students. The aim was to assess how easily the students can be taught a concept that they are unfamiliar with. For this purpose some of the best college teachers in the committee (particularly those who have won teaching awards) were sent to schools. The students happily participated in this exercise, and their sheer enthusiasm and youthful energy was infectious. The students were given hints throughout the interaction.

The second aim was to assess the current level of knowledge, logical skills, and FCS preparedness. The aim was to do this without making the students feel uncomfortable, and to do this in a playful manner such that the performance reducing effects of mental stress can be avoided. The committee was able to very successfully evaluate the logical and analytical abilities of students, correlate them with their board marks and teachers' perceptions.

4.3.2 Results of the Interactions and Subsequent Assessments

Before presenting the results, a word of caution is due. Note that there is a big difference between a student's *current level* and *future potential*. The former represents the learning outcome of the last few years of education, and the latter represents her long-term potential if she is subjected to a more demanding and more scientific curriculum. Just because a student has a low *current level* does not mean that she has a bleak *future potential*. All that this means is that if remedial measures are taken, students can achieve their fullest potential and also perform at the same level as their international peers.

Level 1: Students in level 1 schools were able to merge sorted arrays very easily. Subsequently, they were able to grasp the idea of the divide-and-conquer strategy. However, they faced issues in writing the logic of the algorithm correctly. They made errors when it came to the proper sequencing of the operations. With some hints, all of them were able to correct their programs. Subsequently, the students were asked to figure out merge sort by themselves. Once the divide-and-conquer strategy was explained to them, they were able to figure out the basic merge sort technique.

```
Step 1: Break the arrays into two parts
Step 2: Sort each of the sub-arrays (same problem albeit with smaller inputs)
Step 3: Merge the sorted arrays.
```

Step 2 is the most difficult step, because students require some degree of abstract thinking here. They need to appreciate the idea that we are calling the same sort procedure with smaller inputs. This idea is known as *recursion* in computer science. This is also an example of a divide and conquer strategy where we break a larger problem into two smaller sub-problems. Each of the sub-problems is solved separately by invoking a separate instance of the sort routine.

However, when it came to count the number of operations, the students faced difficulty. They were not able to connect this with the notion of logarithms that they had learnt in classes 9 and 10. After some explanation, most of them got it.

Then we asked them to do some simple problems like write the code to compute x^n given x and n, where n is a natural number. They made errors in the logic (for more about common errors, refer to [Rivers et al., 2016]). They were not able to write a simple iterative program to compute x^n in the first attempt. They needed some coaxing and hints.

The committee wanted them to write these three lines, which are fairly simple:

```
prod = 1;
for (i=0; i<n; i++) prod = prod * x;
printf ("x^n = %d\n",prod);
```

It is true that the children have a lot of potential given their high rate of learning; nevertheless, it is also true that the level of preparedness of particularly class 12 students is not at par with international students who have learnt computer science for 2 years in their 11^{th} and 12^{th} classes.

Level 2: Students in this range had a fairly large amount of diversity. However, the committee found them to also be very good. They were able to grasp the concept of recursion and merge-sort very easily. They had some difficulty in recalling the definition of logarithms and using the notion of a log to find the number of operations in a merge-sort routine.

They also had issues with the logic when it came to merging sorted arrays particularly when they had to figure out if we need one loop (a piece of computer code that performs the same task over and over again albeit with different inputs), or two loops.

The committee members observed that the students were very intelligent; they could grasp concepts very quickly. Furthermore, they were also extremely well mannered, polite, and enthusiastic. Their main difficulty was in writing the logic of the programs. There they were making simple yet avoidable logical errors such as not initializing variables with the correct values, misplacing loops, and not accounting for corner cases.

Similarly, in one more school, students had a fair amount of difficulty in finding the third largest element of an array. This requires some amount of decision making, which is pretty mechanical in nature. They were able to get to the solution with multiple hints after 10 minutes. Particularly for students in class 12 who have been seeing this material for the last one and half years, a much better performance was expected.

Level 3: The students in this category of schools were also extremely keen to learn, and of course all students were very polite, and social. The committee in fact noted that the social skills of students is far better today than what it used to be when they were students. Let us describe a representative interaction.

Students in this category of schools had some conceptual problems from the beginning. Unlike students in the other two classes of schools, they found it hard to merge sorted arrays. They had difficulty in figuring

out the smallest element of the combined array. When the committee members told them that it would be the smaller of the smallest elements of each of the arrays, they found it easier to visualize the situation.

The basic mathematical operation in which they had difficulty is: $min(S_1 \cup S_2) = min(min(S_1), min(S_2))$.

Then they were asked to write a function that computes x^n, where n is a natural number. They made some basic mistakes in initializing the variables. Here is the piece of code that they agreed with:

```
int func (int x, int n) {
    for (i=1; i <= n; i++)
       x = x * i;
    return x;
}
```

They were told that the answer is closer to n factorial. Again this program does have errors. The variable x is not initialized.

However, after explaining a lot of concepts, they found it much easier to understand. It is not that their learning potential is very low, the committee members observed that the main issue is that they don't have an adequately strong mathematical background. This should have developed in their primary and middle school. With a strong mathematical background, they could have easily solved these problems. This is where students in level 1 and 2 schools have an advantage. Their basic background in mathematics and logical thinking is significantly better.

4.3.3 Overall Impressions

The overall impression is that the learning outcomes are not up to the mark for even students from the best schools in the country. This is not just a problem with one region, it is a nationwide phenomenon. The problem is particularly more acute in schools where a lot of underprivileged students study. They definitely need a much better education in mathematics at the primary and secondary levels. This will not just create a good mathematics and CS background, it will help them in other areas also.

The committee correlated these observations with the level of preparedness of students when they take their first semester CS courses in Engineering colleges. The results are very well correlated. The focus on logic and the mathematical aspects of CS is paramount. This is not happening. Unlike ASER studies for primary education, we don't have a lot of data for CS proficiency in schools. The only publicized study that the committee could find was a report [PTI, 2017] in the Hindu conducted by an assessment company called Aspiring Minds. The results indicated that after college 95% of programmers cannot write correct programs, and only 1.4% can write efficient programs. It is true that for this situation a bulk of the responsibility lies with the colleges. However, the sad part is that these colleges are unfortunately training future school teachers.

4.4 Reasons for Poor Performance

The reasons for poor performance are mainly because of some misconceptions and consequent decisions spanning from policy to implementation.

Misconception: Computer science is all about learning a language such as C++ or Java.

Explanation: This is not correct at all. A programming language is a vehicle to express a structured algorithmic concept. The thought process and logical thinking is of utmost importance, not the way it is expressed. Because of this thinking, there is hardly any focus on problem solving in schools. There is significant emphasis on teaching the syntax of languages such as C++ and Java. Students are spending more than 80% of their time learning about which ';' goes where in a program, and which ',' goes where. They are missing the big picture and thus not developing the required logical and analytical skills. Let us explain this another way. Bicycling is a basic skill; whereas, the bicycle having gears is not important for a beginner. If we have a training method where we only teach how to use gears correctly to 5 year olds, they will not learn how to ride a bicycle properly. It is thus important to interpret programming language concepts in

the right perspective and not focus exclusively on the syntax. It is important to note that **programming languages come and go; however basic computational skills remain with a person for the rest of her life**.

Further note that in a typical computer science B.Tech program we roughly have 30-40 courses. Out of these only 1 or 2 are programming courses. The rest of the courses deal with many other aspects of a modern computing system such as hardware, operating systems, networking, graphics, AI, databases, data mining, computational complexity, and discrete math. All of these areas require a core training in formal CS concepts.

Misconception: Because a given programming language has a good future scope, it should be taught.
Explanation: Yes and no. The job of a school is not to make students immediately job ready. Between the 12^{th} class and an IT job, we have 4 years of college. The task of making the child job-ready falls on the college. The role of a school is to give foundational skills. Note that 14-17 year old children are at their intellectual prime. At this age, they can pick up difficult concepts very easily. It is thus best to teach them as many problem solving skills as possible. Given the dynamic nature of the computer science industry, technologies come and go very quickly for many reasons. Sometimes the reason can be as trivial as an executive in a major company simply doesn't like a given language. Given these vicissitudes of life, we cannot assume that a given technology will be alive after 5 years (maybe not even after 2 years). However, the basic principles of programming have remained the same for the last 60 years. Unless there is a significant change in the basic paradigm of computing (very unlikely to happen), these skills are permanent.

Additionally, the logic that just because a language is taught in college in the first year of engineering, it should be taught in school, is also not correct. Different colleges teach different languages in their first semesters, and often the languages change with the instructor. A top college can teach Python one year, Matlab the next year, and Java the subsequent year to first year students. It thus cannot be assumed that a given programming language will be taught in college. Also note that a programming language course in the first semester barely accounts for 1.5% of the total CS B.Tech curriculum and is thus insignificant. It is better to focus on skills that are relevant to all of CS.

Finally, note that the perception of committee members who teach Ist year engineering courses is that the difference between a student who has studied CS in school and a student who has never touched a keyboard is significant to start with. However, after two months, the students are pretty much indistinguishable. One with superior analytical skills ultimately wins the race.

Misconception: It is a good idea to specialize and have many different CS courses/streams.
Explanation: This has resulted in a lot of courses. There is one course for those who want to build desktop and web applications, there is one course for those who want to write simple command line based programs, and one course for those interested in web technologies. Over specializing at a young age is not recommended. It is also not a part of the ACM recommendation or any international curriculum. There is more or less a consensus on this issue, which is that we cannot teach basics of CS, and a very specialized application specific area at the same time. We will end up doing injustice to both the areas. The committee believes in the dictum, **have at the most one or two courses and teach them well.**

Misconception: Students at the end of class 12 cannot produce anything useful.
Explanation: This is also not true. The committee envisions a different kind of project (which all CBSE students have to do in class 12) and lab activities. This vision has been borrowed from foreign curricula where students often do projects with a lot of societal relevance. There is no reason why a student cannot walk up to a neighbourhood shopkeeper, and try to solve some of his problems with her newly acquired CS knowledge. The student should be empowered to do all of this. Moreover, the student should also be able to help herself to do most of the tasks that a modern society requires in the future: calculating EMIs, calculating taxes, maintaining an inventory of items, and do post-retirement financial planning.

Misconception: Computer science is a subject mostly for boys.
Explanation: This is probably the most pernicious misconception of all. It automatically excludes half

the population for no fault of theirs. The committee also noted that some school teachers are sometimes knowingly or unknowingly discriminating against their female students, even though the teachers are female themselves. The committee has heard anecdotal accounts from teachers and students; however, it is not in a position to verify the veracity of these accounts. However, since some of these accounts were in the first person, there is a need to address this issue.

The committee would like to firmly state that such biases and stereotypes are not rooted in science. These are prejudiced and baseless stereotypes. **Women can be as good computer scientists as men, if not better.** A humble request is made to members of the CBSE board and the wider public to dissuade people from harbouring and propagating such stereotypes. At the moment the CEOs of IBM, Oracle, Xerox, and the COO of Facebook are women. Women have been recipients of the highest awards in computer science including the coveted Turing award (equivalent to Nobel prize in computer science). There is ample empirical evidence and there are ample examples to show that women can do as well as men when it comes to programming and logical thinking.

Misconception: A well stocked lab and a computer are absolute necessities for teaching programming.
Explanation: Yes and no. Definitely a computer lab is required for writing and running programs. However, that said and done, having a computer all the time is not a necessity. In fact curricula in New Zealand [Bell et al., 2010, Bell et al., 2014], England [Brown et al., 2014, Jones, 2013], and Finland [Hubwieser et al., 2015b] have fairly large components that do not require a computer. A set of normal household objects are good enough. They can be used to teach very important CS concepts such as selection, iteration, searching and sorting. In fact, in the seventies and eighties one of the favourite questions asked in technology companies was to propose an automated algorithm to tie a shoe lace or a tie. It is ironical to think that some of the best computer scientists in the world today were hired this way! Since FCS is all about logical thinking, it can be done in many real life situations, even when students don't have access to a physical machine.

Nowadays, even a smart phone is a first class computing device. It can be used to write and run programs. Hence, having a computer all the time is not a strict necessity, even though it is desirable most of the time. In the same vein, sometimes not having a computer of any form might be beneficial. Students learn better if they work with simple objects, and achieve some objective like sorting them, by physically swapping the positions of objects.

4.5 Obsolescence of the Current Curriculum

The current CBSE curriculum dates its origin back to the late nineties. It has undeniably served the needs of the nation very well. Nevertheless, it is important to understand that there has been a significant change in the computing landscape ever since. It is thus necessary to ensure that the curriculum at the school level tracks these changes, and is also in line with the curricula in other countries.

After a thorough study of recommendations, international curricula, learning outcomes, and available resources, in the view of the committee these are the issues with the current curriculum:

1. There are three courses in classes 9 and 10, which are very similar. It is not necessary to have three courses that are so alike. Furthermore, there is very little FCS component in these courses. Thus, there is a need to introduce some degree of logical thinking in these courses.

2. In classes 11 and 12, an overwhelming portion of the time is going in teaching the syntax of C++ or Java or Javascript. This is strictly the opposite of what is recommended and currently followed worldwide. The focus on formal CS concepts (FCS) is minimal. This is leading to a very poor learning outcome.

3. Three courses in classes 11 and 12 are not required. The motto should be to teach less, and to teach it well.

4. Roughly 60% of the students today are learning C++. C/C++ is a very good language for scientific computing and embedded systems. It was never designed for general purpose use. This explains why we still don't have a good development environment on Microsoft Windows other than proprietary solutions such as Microsoft tools or tools that clone a Linux environment (Cygwin, MinGW). Students are using an obsolete Turbo C compiler, which was made to run on MS-DOS (technology of the early nineties). It is also outdated in terms of supporting language features, and most students including the disabled have a very serious problem with the interface. There is a need to use a far more modern language with a very user-friendly development environment.

5. The different parts of the C++ based course taken by a majority of the students are not connected. C++, and SQL are taught separately. However, students are never taught how to connect them, and make them run together as a part of a single application. As a result, they never understand the connection between a programming language and a database. There is a need to create one well rounded integrated curriculum without any internal inconsistencies.

6. Learning Java (in the IP course) or PHP/Javascript (in the MWT course) are not the best languages for teaching beginners programming. These are fairly syntax-heavy languages with a lot of boiler plate code (pieces of computer code that need to be added at various points of a program without any alteration). They may confuse beginners, and are more likely to divert attention to unnecessary details. There is a need to keep things simple at the beginning, and ramp the complexity when it comes to logical thinking and problem solving.

7. The IP and MWT courses introduce a very light version of computer science. This is not doing justice to the students who are taking the course because they are not learning the skills that they can carry into the future. They will also find it hard to transfer their skills from one technology to another.

8. The notion of computational efficiency is not being looked at. Students have to know why a given algorithm is fast and another algorithm is slow. This is not being taught, as a result the performance aspects of computing are not being looked at.

9. There is an issue with projects being done by students at the end of class 12. Since students are restricted to only the command line, the number of projects that they can do gets reduced significantly. It is instead a better idea to teach them a language with extensive library support such that they can create expressive applications with graphical interfaces, and possibly animation and sound. Working with such library functions (pre-written functions) is not being emphasized in labs.

10. Students in today's world need to get a much better understanding of cybersafety, and cybercrime. In addition they need to understand the impact of technology on society, and also navigate themselves in a complex world with identity theft, and biometric ids. This requires a curriculum redesign for these topics.

11. The committee looked at the board exam papers of the last few years. They are undoubtedly on the easier side, and could do much more to stimulate critical and computational thinking among students and teachers. The hardest problems barely require a few lines of code, and were deemed by the committee to be fairly easy for passing out class 12 students. The committee has suggested a modified examination pattern (see Section 7.4), where there are questions of increasing difficulty. Roughly 75% of the marks should be accounted for by questions that are in terms of difficulty between easy and medium. 20% of the paper should require significantly more understanding and thinking, and 5% of the paper should be very difficult (only for the top 1%). This gradient will help create a climate of excellence.

5 Main Guiding Principles for Designing a New Curriculum

The main principles in designing a new curriculum are: have a set of few well designed courses that focus on formal CS principles, and rectify the issues pointed out in Section 4.5.

5.1 Major Decisions and their Justification

5.1.1 One Course in Classes 9 and 10

The percentage of students who appeared in the class 10 board exam in 2017 for each of the courses is as follows:

Course	**Percentage of students**
Foundation of Information Technology (FIT)	98.8%
Information and Communication Technology (ICT)	1%
E-publishing and e-office	0.2%

An overwhelming majority (98.8%) of students take the FIT course, and a very small minority take the rest of the courses. Moreover, as pointed out in Section 4.5 there is a significant overlap between the courses. There is per se no reason, to support three separate courses, when their content is roughly the same, and two of the three courses are hardly subscribed to. Secondly, supporting so many courses represents a logistical challenge to CBSE. Resources are better utilized in designing one good course.

Hence, the committee took a decision to discontinue two of the three courses. There will be a single course with the name *Computer Applications*. It will be called *Computer Applications 1* in class 9 and *Computer Applications 2* in class 10. This course will mostly contain what the previous courses used to contain. However, the Computer Applications course will be better organized in terms of content.

Since a need was perceived to introduce some FCS content in the course, the committee decided to use a very simple visual programming language called Scratch in class 10. It has been developed at MIT, and is meant to be used as an educational programming language for beginners. It is purely based on a graphical interface, and is very easy to use. It is meant to introduce FCS concepts in a very gentle way. However, for children with visual impairments, or with difficulty using a mouse, it will be hard to use Scratch. Hence, they can be taught a very light version of Python such that they can pick up similar concepts.

5.1.2 Two Courses in Classes 11 and 12

Let us take a look at similar figures for the computer science courses offered in classes 11 and 12.

Course	**Percentage of students**
Computer Science (CS)	58.2%
Multimedia and Web Technology (MWT)	3.2%
Informatics Practices (IP)	38.6%

Here, also there is an asymmetry. The CS and IP courses are taken by most of the students. A small minority (3.2%) of students take the Multimedia and Web Technology (MWT) course. Just because a course does not have takers is not a strong enough reason by itself to scrap the course. However, it can be one of the reasons particularly if the additional cost of supporting an extra course (exams, centres, paper setting, paper correction) outweighs its benefits.

In this case the committee thought that the costs genuinely outweigh the benefits. Supporting an additional third course did not seem to be a very viable option from the point of view of both education and logistics. When foundational areas are weak, there is a need to work on them first. Furthermore, multimedia

and web technologies have a more vocational component than a theoretical component (the way it is currently taught). In fact students are mostly being taught programming with Javascript and PHP, which are not recommended for beginners. Even for multimedia tools, the committee noted that students are being taught about the features of some commonly available tools. However, this is not dealing with the artistic aspects of multimedia creation. The artistic aspects are difficult skills, and need to be evaluated by an experienced person in a one-on-one setting. In other words, several aspects of a proper multimedia course are very artistic in nature, and thus cannot be evaluated with a theory exam conducted by a central board in a large country such as ours. The course is bound to degenerate to an offering where only the features of any particular application are discussed, and there is no real focus on knowledge and skills. This is exactly what the committee wants to avoid. The committee does not want to compromise on rigour in any of the courses.

Given these reasons, it was decided that the MWT course will be dropped. The CS and IP courses will be retained, and will be upgraded to better and modern versions.

5.1.3 Computer Science 1 and 2

It was decided to keep the same name for the CS (Computer Science) course, and call it Computer Science 1 in class 11, and Computer Science 2 in class 12. However, the committee observed that it is necessary to make some changes to this course mainly to introduce FCS concepts, and rectify some issues in the previous version of the curriculum.

Here are the important decisions taken with respect to this course.

1. Replace C++ with Python (see Section 5.1.5) for detailed justifications.
2. Reduce the amount of content, remove all the obsolete content, streamline the course, and introduce more FCS components.
3. Create one cohesive course, where all the parts are well connected.
4. Have a distinct module that focuses on cyber ethics, social media etiquette, and cyber crime.

Python was a part of the CBSE curriculum till this year (2017), and thus a lot of teachers are trained in using the language; however, for the rest of the school teaching community, this might be a new language if they haven't learnt it before. Nevertheless, given the fact that Python is a very easy language, the committee does not foresee any problems. There are many more reasons for choosing Python and the committee feels after a lot of deliberation that making this choice is essential (detailed justification given in Section 5.1.5) to creating a modern course.

Other than this change, the rest of the changes in terms of content are relatively minor. However, the way that a concept is going to be delivered needs to change significantly. Instead of explaining syntax, the new curriculum emphasizes teaching logic and problem solving. This needs to get reflected in the pattern of teaching and assessment.

5.1.4 IP 1 and 2

The committee looked at the Informatics Practices Course (IP) in detail. This course is mostly taken by students in the Commerce stream (and some in the Humanities stream). At the moment, students are learning Java and SQL (language for interacting with databases). The committee felt that the curriculum was very light when it came to formal CS topics. It was a great idea for its time because it provided an alternative to students who did not want to do a lot of conventional programming, and wanted to do something that is more application-focused. The main need in those days was to create interfaces that are connected with a database (backend). However, these needs are not very relevant today. A lot of the work in designing frontends (command line or graphical views of a software) is significantly easier as of today because of the advances in programming frameworks. Moreover, with much of the computing moving away from the desktop to cloud computers, remote servers, and mobile phones, interface designing has completely changed.

To do justice to this course, it is necessary to teach students how to design interfaces for all kinds of devices – phones, tablets, laptops, desktops, and thin clients. This is too specialized a field, and is better taught in college or learnt on the job. Furthermore, this area involves more of technology, and less of fundamentals. The committee thus felt that from both an educational and technological point of view the current IP course has lost its relevance.

However, at the same time, the committee also took into cognizance the view that having a single course in classes 11 and 12 is not the best option. The committee did agree with this view primarily because computer science by itself has become very broad, and different subsets of it can be carved out for different sections of students with different abilities and aspirations.

Keeping this in mind, the committee visited schools and interacted with Commerce students, who are the main subscribers of the IP course. The committee understood the structure of economics, accounting, and business management courses that are currently being taught in classes 11 and 12. Along with the interactions the committee discussed many ideas with eminent people in academia and industry.

The broad insights that they gathered is that we are in what is called Society 4.0. We started with a hunter-gatherer society (Society 1.0), moved to an agrarian society (Society 2.0), and then an industrial society (Society 3.0). Now, we are in an information age (Society 4.0), where a whole world of information lies at our fingertips. Not just Google and Facebook, a typical small business or enterprise generates a massive amount of *data*. All items of inventory, all transactions, and all communication is computerized, and detailed records are kept. It is often necessary to analyse this data, draw inferences and make predictions.

In other words, *data science* – data analysis, inferencing, statistics, and visualization – has pervaded all areas of society. It is no more restricted to financial institutes and market research companies (refer to the Forbes magazine article [Hugo Moreno, 2017]). Today, even small shops in major cities are maintaining and analysing detailed logs of their transactions. Based on historical patterns they are stocking appropriate amounts of inventory during different times of the year. With the coming of GST in India, the need for automated processing of transactions, invoice creation, and tax computation have increased tremendously. The committee spoke to many shopkeepers in the South Delhi area, and found that many of them are hiring fairly expensive part-time accountants to process their GST claims. They indicated that they would find it very profitable to have more automated techniques and would prefer accountants who can process their claims more quickly using computer programs.

Given the ubiquity of data analytics and processing in today's world, the committee decided to introduce a module on data science for students in classes 11 and 12 (as a part of the IP course). The module is on the lines of similar offerings all over the world (refer to highly popular courses on edX, Udacity, and Coursera). Furthermore, data science courses have started in middle school and high school in the US. The committee took a look at the efforts made by the University of California at Los Angeles (UCLA) in teaching data science in schools [Gould et al.,] starting from 2014, and the course developed by Bloomberg and academics from Brown University([Shriram Krishnamurthi and Emmanuel Schanzer, 2017]). It was nice to learn that the California university system recognizes [Amelia McNamara, 2014] the data science course taught in schools. The course has been approved as a "C" course, which means that it will count for the three required years of mathematics training. It is thus an integral part of the college preparatory curriculum as per their rules.

The upgraded IP course with a module in data science will have an overlap with the Computer Science (1 and 2) course. However, an effort has been made to minimize the overlap.

For example, some of the advanced features of programming such as multi-dimensional arrays, sorting, recursion, and complex debugging will not be there along with the modules on computer hardware and networks. The new module on data science will introduce methods to analyse, process, and visualize data.

Furthermore, in lieu of computer networks, the committee was of the view that a module on basic software engineering should be there. It will introduce students to the business processes that are involved in large enterprise software starting from planning, and requirements gathering to execution, and validation at the customer site. Students in the commerce stream will particularly need this knowledge if they wish to make a career in the IT industry particularly the consulting business. Even otherwise, understanding the basic processes in the IT industry will prove to be helpful to students in diverse branches of business and

management studies.

Furthermore, for humanities students the committee increased the degree of depth in the module on society, law, and ethics. They introduced more concepts in class 12 particularly on net neutrality, internet addiction, reinforcement while browsing the web, use of new media, online campaigns, smart social mobs and contemporary case studies – Wikileaks, Arab Spring, and Bitcoin. These changes were recommended by Professors in Social Science.

This is a futuristic course that will serve the needs of those who expect to work with a lot of data and information starting from businesses, to financial institutions, to large enterprises, very well.

Akin to the Computer Science 1 and 2 courses, the upgraded IP 1 and 2 courses will also be taught in Python. The reasons are mentioned in Section 5.1.5.

5.1.5 C++/Java vs Python

The committee deliberated on this issue at length. They were aware of the fact that C++ and Java are being taught for the last 20 years. They were also aware of the fact that the previous curriculum committee chaired by Prof. Neelima Gupta (from Delhi University) had recommended Python in 2013. This recommendation had been accepted by CBSE; however, an option to keep the earlier C++ based course was provided to ease the transition. However, the C++ based course was not discontinued. Subsequently, the Python based option was discontinued in 2017 citing the logistical challenges involved in supporting two courses: one based on C++ and one based on Python. However, in spite of the chequered history of introducing multiple languages, the committee after thorough discussion, and in consultation with renowned experts in the field was overwhelmingly in favour of introducing Python as the programming language of choice. The technical experts in the committee and the set of external experts who were consulted have an absolute unanimity in this regard. This decision is in line with worldwide trends for the first programming language [Guo, 2015, Bell et al., 2010, Baron et al., 2014].

There are many other reasons for choosing Python over C++ and Java. They are as follows:

1. At the level of classes 11 and 12, developing the ability to logically solve problems irrespective of the programming language is the most important. Most international curricula are stressing on programmatic and algorithmic thinking. A deep knowledge of programming languages is not a primary concern. It can be dealt with later. It is best to choose a language that is not syntax heavy, yet is very popular. Python falls in this category.

   It was observed that currently students in schools are spending more than 80% of their time learning syntax. There is hardly any time left to solve logical problems or learn about other areas in computer science, and that has led to a very poor learning outcome. It is thus wise to reduce the burden of learning the syntax by avoiding syntax-heavy languages such as C++ and Java.

2. Last 30 years of research in programming languages has brought us to a point where almost all low level aspects of the underlying architecture and the memory system have been abstracted and virtualised. These are highly desirable features and should not be avoided because they allow the programmer to focus more on the logic rather than on hardware features. Low level features are better taught in a course on computer architecture rather than in an introductory course in high school. It is thus wiser to opt for Python that embodies a lot of this research; in specific, it has support for built in memory management and garbage collection. It also natively supports high level data types such as lists and hashtables along with iterators. All of these are very useful features and make programs very simple. The committee also noted that most non-semantic bugs in serial programs are memory bugs, which can be avoided and tracked much better in Python. Pointers and manual memory management in C++ cause far more problems than they actually solve; hence, Java also got rid of them way back in 1995.

3. A standard argument for advocating C++ and Java in schools is that they help students get an advantage if they join an engineering or science discipline in college. This is not true, and is also

not something that is very important. A 4-year engineering course typically has 40 courses, and a 3-year B.Sc. in computer science has 30 courses. An extra advantage in the first course is barely statistically significant. Moreover, most engineering colleges abroad have shifted to non-C++ languages such as Python and Matlab (article in ACM Communications by Guo [Guo, 2015]) and so have many engineering colleges in India. Even if the student does get an advantage in the first programming course, in a CS program, it is still not very important. There are 20-25 more CS courses that the student needs to take. They are not based on teaching programming, and they require a student to have mature logical thinking. In this regard, teaching FCS concepts in schools is a much better bet instead of focussing on the programming language.

4. Both C++ and Java are fairly heavy on boilerplate code (code that should be part of a program irrespective of the logic). In C++ we need to have include statements, and define a main function. In Java every program needs to be object oriented, and needs to have a lot of additional lines and keywords. Python in comparison does not need any boilerplate code. This simplifies programming significantly.

5. There is a serious issue when it comes to the availability of C++ compilers on the most popular OS – Windows. Unless users have access to Microsoft VC++ (natively or with LLVM Clang), they are limited to MinGW (TDM-GCC) or Cygwin on Windows. Microsoft tools are proprietary and Cygwin is hard to install (sometimes very hard), whereas TDM-GCC is much easier and can also be packaged with the CodeBlocks IDE. However, all versions of GCC on windows use a minimalist version of the GNU system (MinGW) and have limited support for advanced features such as process and file management. Moreover, MinGW is not fully POSIX compliant unlike Cygwin – all important POSIX features are not necessarily supported. Lack of compliance causes issues while writing programs, and doing projects that have a lot of features. At least at the school level, there is a need for a framework that is extremely robust.

   There are other free software such as PellesC, which is yet to become very popular and mature; hence, the committee felt that the compilation support for Python is far more advanced and mature on almost all platforms including mobile phones. Furthermore, students don't like the arcane and obsolete TurboC interface that they use (based on MS-DOS); they want to use an editor of their choice. Also, the fact that the easy to use compilers for C++ on Windows don't exist means that clearly there is no demand for it at the school level, in geographies where people can't afford to buy Microsoft software. The committee thoroughly discourages the use of pirated software, and does not want students to download pirated copies of Microsoft Visual Studio from torrent sites.

6. Most schools in India use TurboC. It supports a version of C++ that existed in the mid nineties. There have been no upgrades to the software henceforth. It is high time that students move away from obsolete technology. Moreover, for students with disabilities it is very important to use editors of their choice such that they can avail the right accessibility features.

7. The issue of library support (advanced application specific features such as graphical elements) is also very important. A student using C++ is restricted to the command line unless she wants to use Boost, GTK, QT, or MSVC libraries. These are not platform independent, are often proprietary, are fairly hard to use at an introductory level, and it is very hard to do worthwhile school projects. It does not look like that any major technological companies are investing in easy-to-use cross-platform C++ libraries. In comparison, Python has a great advantage in terms of library support. Everything from web servers to graphical interfaces to parallel programs can be written with a few lines of code. If a student wants she can at a later date develop large real-world applications. It will be very hard to do so with a purely console (command line) based setup.

8. C++ and Java have very little support for working with data. This is why most companies that deal with a lot of data use SAS, R, or Python. SAS is a proprietary language. The committee chose Python to ensure that the same pool of teachers can teach both the courses if there is a need. There is no

need to introduce an additional language – R. Python has extensive support for data analysis, data visualization, and statistics. It is very easy to do complex data analysis tasks and then plot the results. Moreover, Python is recognised as a first class language for data science, and thus this choice was made.

9. It is important to understand that every language was designed for a certain purpose. C and C++ are great for embedded systems, scientific computing, systems software, action based games, and video editors. Java is great for enterprise software. In comparison, Python from the outset was designed for general purpose usage. It was thus much easier to use, and Python programs are often very compact (2-3X shorter than C++/Java programs).

5.2 Making the Curriculum Friendly to Children with Special Needs

Since the beginning of this exercise, an explicit aim of the course was to make this course friendly to children with special needs. The committee is of the view that even if one child in a school has a special need, then a very sincere effort should be made to educate the child. A lack of resources, or a lack of training, is not an acceptable excuse in today's world. All of these can be acquired fairly easily. It is the job of the school to summon the resources, seek help from the respective governments, and create an appropriate learning environment for the child. The computer science community thoroughly believes in the *Ubuntu* philosophy, which in its native Zulu refers to a very unique characterization of humanity. In the words of the great Nelson Mandela it can be understood with the following example, "A traveller through a country would stop at a village and he didn't have to ask for food or for water." It also means, "I am what I am because of who we all are" (from ubuntu.com).

The committee went through the schedule of disabilities in The Rights of Persons with Disabilities Act, 2016. The following broad classes of disabilities were noted:

- Locomotor disabilities: severe deformities, polio, leprosy, cerebral palsy
- Hearing and speech disabilities: hearing impairment, speech aphasia
- Cognitive impairment: specific learning deficits (Dyslexia, Dyscalculia), Down's syndrome, Autism
- Vision impairment: low vision, blindness

Out of these classes, hearing and speech impairment, is not an issue for teaching computer science unless we are dealing with sound. However, audio processing is not particularly important in the proposed curriculum structure. Hence, this disability will not cause any issues with the curriculum. Let us go through the rest of the disability categories.

5.2.1 Locomotor Disabilities

A lot of children in this category suffer from some form of permanent deformity, polio or cerebral palsy (brain damage before or at the time of birth). The issue is that not all locomotor disabilities are the same. If the child can type and use a mouse, then there are no issues. Some children can operate a mouse and keyboard with their hands like non-disabled children, and a lot of children can amazingly operate these devices with their feet. However, a lot of children in this category are both keyboard disabled and mouse disabled. The committee still did not want to exclude these children from a good and effective CS education.

The committee thus looked at research papers on the web, and talked to eminent experts in the area such as Prof. Anupam Basu from IIT Kharagpur. Prof. Basu has at least 20 years of experience in devising technologies for children with cerebral palsy and blindness. In particular, the committee was influenced by the following research papers [Man and Wong, 2007, Davies et al., 2010]. The researchers observe that children with severe cerebral palsy find it hard to use a mouse. However, it is possible to design assistive devices for them. For example, some children can type with a virtual keyboard where keys are selected with a specially designed joystick. In such devices moving the joystick selects the key, and then there is a method

to press the key by having a special button in the joystick. Note that the great physicist, Prof. Stephen Hawking, can type by basically moving one muscle on his cheek.

The committee further noted that particularly for Python programming, we don't need all the characters that a standard keyboard provides. To study this further, a Ph.D student at IIT Delhi, Mr. Shubhankar Suman Singh, wrote a program to analyse the frequency of characters in large C++, Java, and Python programs. He found that with roughly 15 keys, we can write 90% of today's Python programs. This number increases to roughly 25 keys for C++ and Java. The committee thus concluded that if a virtual keyboard is designed with roughly 12-15 keys (or maybe slightly less), we can easily write programs. It is not necessary to have variable names with perfect spellings. We can easily skip some letters. This will reduce the number of letters that we require to write a program.

Now, to use a virtual keyboard to write a program, we need a fair amount of support:

1. Sophisticated virtual keyboard software.
2. A joystick that is specific to the needs of the child.
3. A programming editor that can be interfaced with the virtual keyboard, and does not require any mouse movements.

None of these are difficult tasks by themselves. However, there are several operational challenges. First, we need a compassionate school that is willing to look into the needs of a child and refer the case (if needed) to experts. Then, the software and hardware need to be procured. Many such assistive devices are not manufactured locally, they need to be imported. This increases the cost. We need to have some method where the school or the government pays for these devices, if the student and his family are not in a position to afford them. If these problems are sorted out hopefully with assistance from the school and the respective government, then most children with locomotor issues will be able to make good progress in the CS course.

5.2.2 Learning Deficits, Down's Syndrome and Autism

There can be learning deficits of many kinds: Dyscalculia (difficulty with math), Dysgraphia (difficulty with writing), and Dyslexia (difficulty with reading). Since CS subjects are introduced in class 9, by implication, it can be assumed that the student has gone through grades 1 to 8. He/she has some understanding of the curricula in classes 1 to 8. This means that irrespective of the nature of the learning deficit, it is not extremely severe.

For children with specific learning difficulties, the committee was influenced by the the Ph.D thesis of Dr. Matthew Taylor [Taylor, 2017]. His thesis indicates that it is possible to teach coding to even younger students (before class 5) with the aforementioned disabilities. However, there is a need to change the approach. The following techniques should be tried.

1. Design experiments that have a tactile feel such as programming a robot. Students learn better.
2. Have smaller classes, or preferably slot some time for one-on-one interaction.
3. Introduce all the concepts in a very structured manner. This is where introducing FCS concepts will help because they are very well defined.
4. For each concept, provide a lot of examples.
5. Give more time, and provide positive feedback.

The authors have observed positive results with children in this category, including a cohort of children with Down's syndrome. However, to get the desired results, it is necessary to put in much more effort, and also take the pains to structure the teaching process significantly.

Another disorder, which falls in this category and is fairly prevalent is Autism. Autism is primarily characterized at younger ages with problems in communication, repetitive movements, lack of social contact,

and stereotypical movements. Even though these are some of the outward symptoms, Autism is associated with some learning deficits, which can possibly cause some of the outwardly symptoms [Sinha et al., 2014]. Autism is increasingly being characterized as a disorder of prediction. This means that individuals with Autism are not able to predict how the world around them will be in the immediate future based on the present. As a result, they live in a magical world, where everything seems new and unpredictable to them. The hypothesis in this school of thought is that because children find the world so overwhelming, they engage in repetitive behaviours that help calm them down. This hypothesis is gaining traction and has been used as a basis by numerous researchers to explain the traits of Autism.

The committee consulted top neuroscience researchers. They indicated that children with Autism find it hard to do things that are new or are open ended. They also have issues with abstract thinking, generalization, and attention deployment.

Here are their suggestions:

1. Introduce the topics one at a time so that attention and learning loads are reduced.
2. Present the information through multiple concrete examples. The examples together should span several of the real-world settings that the child is likely to encounter in the real world.
3. When teaching presentation software, make sure the children are instructed to first create an outline that can be used to guide the specific slides included in the presentation. Children with Autism have challenges in sequencing.
4. Use several examples that are drawn from non-social settings when presenting the teaching material.
5. The well-specified rules of programming language will be helpful for children with Autism.
6. Avoid open ended projects such as web designing, and instead focus on teaching programming and introduce it with several structured learning tasks.

The advice from experts is that the well-specified rules taught in the Computer Science and IP courses should be helpful to children with Autism. The committee had discussions with parents of children with Autism. Their feedback was very similar. Prof. Prabhu Rachkonda who works with children with Autism further mentioned that since children with Autism have an extraordinary eye for detail, they will find it easy to write relatively bug-free programs.

The committee also read a highly encouraging news article published in the Economic Times on Saturday (November 25^{th}) [Rica Bhattacharya, 2017] that mentioned that major tech. companies such as JP Morgan, Dell, Capgemini and Cisco are hiring a lot of people with Autism in India. Given their attention to detail, they are proving to be invaluable assets in software testing, research and proof-reading.

To summarize, the committee believes that by making the positive changes as suggested by eminent people in the area, it is possible to get favourable learning outcomes for children with cognitive and neurological problems.

5.2.3 Blindness and Low Vision

This is probably an area that has been worked on the most by researchers. Almost all operating systems and office suites (word processor, spreadsheet, presentation software) have accessibility features. For example, Microsoft has Narrator and Cortana, which blind people can use to navigate their way through a set of windows, and achieve essential tasks. Similarly, Ubuntu systems have Orca and onBoard. There is a also a version of Linux called BLinux especially meant for the blind. In addition, office software have a host of accessibility features that can be easily activated. These features are present in both Microsoft Office and open source offerings such as Apache OpenOffice and LibreOffice.

Furthermore, it is easy to connect these systems with Braille keyboards, monitors, and printers. These systems are very well supported as of today, and the technology is mature. Along with Braille displays, there are many options available for synthetic speech. Dr. Ramana Polavarapu, who is a part of the committee

is blind. He indicates that blind students should have no significant issues in programming with Python as long as they get to use an editor of their choice.

There are numerous examples of very established blind programmers including Dr. Ramana himself. They use all kinds of frameworks and languages. Hence, blindness per se does not preclude programming. We do need teachers who are sensitive to the needs of the blind, and who have a good knowledge of the tools that they need. In the view of the committee this is not a major hindrance, and blind children should be happily allowed to participate in all programming activities and CS courses.

The only issue is that the Scratch programming language (introduced in classes 9 and 10) is unsuitable for blind students, or students who are mouse-disabled. It does not have accessibility features. The committee deliberated on this issue at length, and was not able to find an equivalent software environment for students in these categories. Subsequently, it was decided that students with a significant vision impairment, or students who cannot use a mouse, will be taught a very basic version of Python. The rest of the students will use Scratch. It might be concerning to some that students with impaired vision or impaired locomotor abilities are getting a head start in terms of Python. They are learning a bit of it in classes 9 and 10, and then they are learning full scale Python programming in classes 11 and 12. However, we should note that students in this category who are unable to use Scratch, are suffering from a serious disability, and it is our duty to help them pick up all the skills that they need to be good programmers. In addition non-disabled children are picking up similar skills while programming with Scratch. Hence, the committee feels that things even out.

6 Curricular Recommendations

6.1 Computer Applications - 1 (Class 9)

6.1.1 Prerequisites

No background in computer science is required.

6.1.2 Learning Outcomes

1. Familiarity with basics of computers.
2. Ability to navigate the file system.
3. Create and edit rich text documents, spreadsheets, and presentations.
4. Perform basic data manipulation using spreadsheets.
5. Use Indian languages in documents.
6. Send and receive emails, follow email etiquette, and communicate over the internet.
7. Create and upload videos.
8. Safe and correct usage of websites, social networks, chat sites, and email.

6.1.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Basics of Information Technology	5
2.	Cybersafety	10
3.	Office Tools	5
4.	Scratch/Python	10
5.	Lab Exercises	70
	Total	100

6.1.4 Unit 1: Basics of Information Technology

- Familiarity with the basics of computers: design of computers, and overview of communication technologies
- Computer Systems: characteristics of a computer, components of a computer system – CPU, memory, storage devices and I/O devices
- Memory: primary (RAM and ROM) and secondary memory
- Storage devices: hard disk, CD ROM, DVD, pen/flash Drive, memory stick
- I/O devices: keyboard, mouse, monitor, printer, scanner, web camera
- Types of software: system software (operating systems), application software, mobile applications
- Operating systems: kernel, device drivers, and file systems (very basic idea)
- Computer networking: wired/wireless communication, common protocols: wifi, bluetooth, cloud computers (private/public)
- Multimedia: images, audio, video, animation
- Chat sites, and social networks.

6.1.5 Unit 2: Cyber-safety

- Safely browsing the web and using social networks: identity protection, proper usage of passwords, privacy, confidentiality of information, cyber stalking, reporting cybercrimes
- Safely accessing websites: viruses and malware.

6.1.6 Unit 3: Office tools

- Introduction to a word processor: create and save a document.
- Edit and format text: text style (B, I, U), font type, font size, text colour, alignment of text. Format paragraphs with line and/or paragraph spacing. Add headers and footers, numbering pages, grammar and spell check utilities, subscript and superscript, insert symbols, use print preview, and print a document.
- Insert pictures, change the page setting, add bullets and numbering, borders and shading, and insert tables – insert/delete rows and columns, merge and split cells.
- Use auto-format, track changes, review comments, use of drawing tools, shapes and mathematical symbols.
- Presentation tool: understand the concept of slide shows, basic elements of a slide, different types of slide layouts, create and save a presentation, and learn about the different views of a slide set – normal view, slide sorter view and handouts.
- Edit and format a slide: add titles, subtitles, text, background, watermark, headers and footers, and slide numbers.
- Insert pictures from files, create animations, add sound effects, and rehearse timings.
- Spreadsheets: concept of a worksheet and a workbook, create and save a worksheet.

- Working with a spreadsheet: enter numbers, text, date/time, series using auto fill; edit and format a worksheet including changing the colour, size, font, alignment of text; insert and delete cells, rows and columns. Enter a formula using the operators(+,-,*,/), refer to cells, and print a worksheet.
- Use simple statistical functions: SUM(), AVERAGE(), MAX(), MIN(), IF() (without compound statements); embed charts of various types: line, pie, scatter, bar and area in a worksheet.

6.1.7 Unit 4: Scratch or Python

Alternative 1: Educational programming language – Scratch

- Introduction to scratch.
- Drag and drop commands, creating simple scripts, repeating blocks of commands.
- Discuss x-y plane, create scripts to move the cat (Scratch mascot).
- Create a script to draw diagrams using the pen feature.

Alternative 2: Python – (provided an option to children with special needs)

- Introduction to Python
- A simple "Hello-World" program
- Running a python program
- The notion of data-types and variables: integer, float, string
- Arithmetic operations: +, -, *, /

6.1.8 Lab Exercises

- Basic I/O devices: use the mouse and keyboard, draw a figure.
- Working with the operating system: Navigation the file system using a mouse and keyboard, and then doing the same with shell commands.
- Word processing: create a text document, create a letter, report, and greeting card.
- Create a text document with figures in it. It should describe a concept taught in another course.
- Discuss the following in a text document about the basic organisation of a computer: CPU, memory, input/output devices, hard disk.
- Create a text document in an Indian language other than English.
- Create a presentation.
- Create a presentation with animation.
- Create and edit existing images, and then include them in a presentation.
- Animate pictures and text with sound effects in a presentation
- Create a simple spreadsheet and perform the following operations: min, max, sum, and average.
- Create different types of charts using a spreadsheet: line, bar, and pie.
- Send an email to your friends. Attach some documents that you have prepared earlier. Put some friend in the CC and BCC list. Interact with friends to find out who was in the BCC list.

- Do an online chat with multiple friends. Transmit documents using the chat platform.
- Create a video and upload it on Youtube.
- Write basic Scratch/Python programs.

6.2 Computer Applications - 2 (Class 10)

6.2.1 Prerequisites

Computer Applications - 1

6.2.2 Learning Outcomes

1. Create a simple website
2. Embed images, audio and video in an HTML page
3. Use style sheets to beautify the web pages.
4. Write iterative programs with Scratch/Python.
5. Interface a web site with a web server and record the details of a user's request.
6. Knowledge of basic cyberethics

6.2.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Networking	10
2.	HTML	25
3.	Cyberethics	5
4.	Scratch/Python Practical	30
4.	HTML Practical	30
	Total	100

6.2.4 Unit 1: Networking

- Internet: world wide web, web servers, web clients, web sites, web Pages, web browsers, blogs, news groups, HTML, web address, e-mail address, downloading and uploading files from a remote site. Internet protocols: HTTP, HTTPS. Remote login and file transfer protocols: SSH, SFTP, FTP, SCP, TELNET.
- Services available on the internet: information retrieval, locating sites using search engines and finding people on the net;
- Web services: chat, email, video conferencing, e-Learning, e-Banking, e-Shopping, e-Reservation, e-Governance, e-Groups, social networking.
- Mobile technologies: SMS, MMS, 3G, 4G.

6.2.5 Unit 2: HTML

- Introduction to web page designing using HTML: create and save an HTML document, access a web page using a web browser.
- HTML tags: *html*, *head*, *title*, *body*, *br* (break), *hr(horizontal rule)*, inserting comments, *h1..h6 (heading)*, *p (paragraph)*, *b (bold)*, *i (italics)*, *u (underline)*, *ul (unordered list)*, *ol (ordered list)*, and *li (list item)*. Description lists: *dl*, *dt* and *dd*.
- Insert images: *img* (attributes: *src*, *width*, *height*, *alt*), *sup(super script)*, *sub(subscript)*.
- Create table using the tags: *table, tr, th, td, rowspan, colspan*
- Links: significance of linking, anchor element (attributes: *href, mailto*), targets.
- Cascading style sheets: color, background-color, border-style, margin, height, width, outline, font (family, style, size), align, float.

6.2.6 Unit 3: Cyberethics

- E-commerce: privacy, fraud, secure data transmission
- Intellectual property rights, plagiarism, and digital property rights
- Software licenses and the open source software movement
- Freedom of information and the digital divide

6.2.7 Unit 4: Scratch or Python (Theory and Practical)

Alternative 1: Scratch

- Revision of the basics of Scratch
- Tempo, variables, and events
- Coordinates and conditionals
- Drawing with iteration
- Update variables repeatedly, iterative development, ask and answer blocks
- Create games, animated images, stories and songs

Alternative 2: Python (for children with special needs)

- Revision of Python basics
- Conditionals: if, if-else statements
- Loops: for, while (e.g., sum of first 10 natural numbers)
- Practice simple programs

6.2.8 Lab Exercises

- Create static web pages.
- Use style sheets to enforce a format in an HTML page (CSS).
- Embed pictures, audio and videos in an HTML page.
- Add tables and frames in an HTML page.
- Decorate web pages using graphical elements.
- Create a website using several webpages. Students may use any open source or proprietary tool.
- Work with HTML forms: text box, radio buttons, checkbox, password, list, combo box.
- Install a web server (Django) and send a GET/POST request to the server. The server will append the fields in the request to a CSV file. The code of the server will be downloadable from CBSE's website.
- Open this CSV file using a spreadsheet.
- Write a blog using HTML pages discussing: viruses, malware spam and antiviruses
- Do a project using HTML forms and a Django web server (strictly NO dbms).
- Create a web page discussing plagiarism. List some reported cases of plagiarism and the consequent punishment meted out. Explain the nature of the punishment in different countries as per their IP laws.

6.3 Computer Science - 1 (Class 11)

6.3.1 Prerequisites

Since a lot of students join CBSE schools from schools run by a state board, we are not assuming any pre-requisites for this course other than basic mathematical skills. However, it will be helpful if the student has a basic knowledge of Computer Applications 1 and 2.

6.3.2 Learning Outcomes

1. Develop basic computational thinking. Learn how to reason with variables, state transitions, conditionals, and iteration.
2. Understand the notion of data types, and higher order data structures such as lists, tuples, and dictionaries.
3. Appreciate the notion of an algorithm, and understand its structure, including how algorithms handle corner cases.
4. Develop a basic understanding of computer systems – architecture, OS, mobile and cloud computing.
5. Learn basic SQL programming.
6. Learn all about cybersafety.

6.3.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Programming and Computational Thinking - 1	35
2.	Computer Systems and Organisation	10
3.	Data Management - 1	15
4.	Society, Law and Ethics - 1	10
5.	Practicals	30
	Total	100

6.3.4 Unit 1: Programming and Computational Thinking (PCT-1) (80 Theory + 70 Practical)

- Familiarization with the basics of Python programming: a simple "hello world" program, process of writing a program, running it, and print statements; simple data-types: integer, float, string
- Introduce the notion of a variable, and methods to manipulate it (concept of l-value and r-value even if not taught explicitly)
- Knowledge of data types and operators: accepting input from the console, assignment statement, expressions, operators and their precedence.
- Conditional statements: if, if-else, if-elif-else; simple programs: e.g: absolute value, sort 3 numbers, divisibility.
- Notion of iterative computation and control flow: for, while, flowcharts, decision trees and pseudo code; write a lot of programs: interest calculation, primality testing, factorials.
- Idea of debugging: errors and exceptions; debugging: pdb, break points.
- Lists, tuples and dictionary: finding the maximum, minimum, mean; linear search on list/tuple of numbers, and counting the frequency of elements in a list using a dictionary. Introduce the notion of accessing elements in a collection using numbers and names.
- Sorting algorithm: bubble and insertion sort – count the number of operations while sorting.
- Strings: compare, concat, substring; notion of states and transitions using state transition diagrams.

6.3.5 Unit 2: Computer Systems and Organisation (CSO) (20 Theory + 6 Practical)

- Basic computer organisation: description of a computer system and mobile system, CPU, memory, hard disk, I/O, battery, power.
- Types of software: application, OS, utility, libraries.
- Language of Bits: bit, byte, MB, GB, TB, PB.
- Boolean logic: OR, AND, NAND, NOR, XOR, NOT, truth tables, DeMorgan's laws
- Information representation: numbers in base 2, 8, 16, unsigned integers, binary addition
- Strings: ASCII, UTF8, UTF32, ISCII (Indian script code)
- Execution of a program: basic flow of compilation: program $\rightarrow$ binary $\rightarrow$ execution
- Interpreters (process one line at a time), difference between a compiler and an interpreter
- Running a program: Notion of an operating system, how an operating system runs a program, idea of loading, operating system as a resource manager.
- Concept of cloud computers, cloud storage (public/private), and brief introduction to parallel computing.

6.3.6 Unit 3: Data Management (DM-1) (30 Theory+ 24 Practical)

- Relational databases: idea of a database and the need for it, relations, keys, primary key, foreign key; use SQL commands to create a table, keys, foreign keys; insert/delete an entry, delete a table.
- SQL commands: select, project, and join; indexes, and a lot of in-class practice.
- Basics of NoSQL databases – MongoDB.

6.3.7 Unit 4: Society, Law and Ethics (SLE-1) – Cybersafety (10 Theory)

- Cybersafety: safely browsing the web, identity protection, confidentiality, social networks, cyber trolls and bullying
- Appropriate usage of social networks: spread of rumors, and common social networking sites (Twitter, LinkedIn, Facebook) and specific usage rules.
- Safely accessing web sites: adware, malware, viruses, Trojans
- Safely communicating data: secure connections, eavesdropping, phishing and identity verification.

6.3.8 Practical

S.No	Lab Task	Marks
1.	Programming in Python	20
2.	Data Management	10
	Total	30

1. **Programming in Python:** At least the following Python concepts should be covered in the lab sessions: expressions, conditionals, loops, list, dictionary, and strings. The following are some representative lab assignments.
   - Find the largest and smallest numbers in a list.
   - Find the third largest number in a list.
   - Test for primality.
   - Find whether a string is a palindrome or not.
   - Given two integers x and n, compute x^n.
   - Compute the greatest common divisor and the least common multiple of two integers.
   - Test if a number is equal to the sum of the cubes of its digits. Find the smallest and largest such numbers.
2. **Data Management: SQL Commands** At least the following SQL commands should be covered during the labs: create, insert, delete, select, join. The following are some representative assignments.
   - Create a student table with the student id, name, and marks as attributes where the student id is the primary key.
   - Insert the details of a new student in the above table.
   - Delete the details of a particular student in the above table.
   - Use the select command to get the details of the students with marks more than 80.
   - Create a new table (name, date of birth) by joining two tables (student id, name) and (student id, date of birth).
   - Create a new table (orderID, customerName, orderDate) by joining two tables (orderID, customerID, orderDate) and (customerID, customerName, contactName, country).

6.4 Computer Science - 2 (Class 12)

6.4.1 Prerequisites

Computer Science - 1

6.4.2 Learning Outcomes

1. Understand the concept of functions and *recursion*.
2. Learn how to create and use Python libraries.
3. Learn file handling.
4. Learn about the concept of efficiency in algorithms and computing in general.
5. Learn basic data structures: lists, stacks, and queues.
6. Get a basic understanding of computer networks: network stack, basic network hardware, basic protocols, and basic tools.
7. Connect a Python program with a SQL database, and learn aggregation functions in SQL.
8. Have a clear understanding of cyberethics and cybercrime. Understand the value of technology in societies, gender and disability issues, and the technology behind biometric ids.

6.4.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Programming and Computational Thinking - 2	30
2.	Computer Networks	15
3.	Data Management - 2	15
4.	Society, Law and Ethics - 2	10
5.	Practicals	30
	Total	100

6.4.4 Unit 1: Programming and Computational Thinking (PCT-2) (80 Theory + 70 Practical)

- Revision of the basics of Python
- Functions: scope, parameter passing, mutable/immutable properties of data objects, pass arrays to functions, return values, functions using libraries: mathematical, and string functions.
- File handling: open and close a file, read, write, and append to a file, standard input, output, and error streams, relative and absolute paths.
- Using Python libraries: create and import Python libraries
- Recursion: simple algorithms with recursion: factorial, Fibonacci numbers; recursion on arrays: binary search
- Idea of efficiency: performance defined as inversely proportional to the wall clock time, count the number of operations a piece of code is performing, and measure the time taken by a program. Example: take two different programs for the same problem, and understand how the efficient one takes less time.
- Data visualization using Pyplot: line chart, pie chart, and bar chart.
- Data-structures: lists, stacks, queues.

6.4.5 Unit 2: Computer Networks (CN) (30 Theory + 10 Practical)

- Structure of a network: Types of networks: local area and wide area (web and internet), new technologies such as cloud and IoT, public vs private cloud, wired and wireless networks; concept of a client and server.
- Network devices such as a NIC, switch, hub, router, and access point.
- Network stack: amplitude and frequency modulation, collision in wireless networks, error checking, and the notion of a MAC address, main idea of routing. IP addresses: (v4 and v6), routing table, router, DNS, and web URLs, TCP: basic idea of retransmission, and rate modulation when there is congestion (analogy to a road network), Protocols: 2G, 3G, 4G, WiFi. What makes a protocol have a higher bandwidth?
- Basic network tools: traceroute, ping, ipconfig, nslookup, whois, speed-test.
- Application layer: HTTP (basic idea), working of email, secure communication: encryption and certificates (HTTPS), network applications: remote desktop, remote login, HTTP, FTP, SCP, SSH, POP/IMAP, SMTP, VoIP, NFC.

6.4.6 Unit 3: Data Management (DM-2) (20 Theory + 20 Practical)

- Write a minimal Django based web application that parses a GET and POST request, and writes the fields to a file – flat file and CSV file.
- Interface Python with an SQL database
- SQL commands: aggregation functions, having, group by, order by.

6.4.7 Unit 4: Society, Law and Ethics (SLE-2) (10 Theory)

- Intellectual property rights, plagiarism, digital rights management, and licensing (Creative Commons, GPL and Apache), open source, open data, privacy.
- Privacy laws, fraud; cyber crime – phishing, illegal downloads, child pornography, scams; cyber forensics, IT Act, 2000.
- Technology and society: understanding of societal issues and cultural changes induced by technology.
- E-waste management: proper disposal of used electronic gadgets.
- Identity theft, unique ids, and biometrics.
- Gender and disability issues while teaching and using computers.

6.4.8 Practical

S.No	Lab Task	Marks
1.	Programming in Python	10
2.	Data management (SQL+web-server)	5
3.	Project work	15
	Total	30

Some sample lab assignments are as follows:

1. **Programming in Python:**
   - Recursively find the factorial of a natural number.

- Read a file line by line and print it.
- Remove all the lines that contain the character 'a' in a file and write it to another file.
- Write a Python function $sin(x, n)$ to calculate the value of sin(x) using its Taylor series expansion up to n terms. Compare the values of $sin(x)$ for different values of n with the correct value.
- Write a random number generator that generates random numbers between 1 and 6 (simulates a dice).
- Write a recursive code to find the sum of all elements of a list.
- Write a recursive code to compute the n^{th} Fibonacci number.
- Write a Python program to implement a stack and queue using a list data-structure.
- Write a recursive Python program to test if a string is a palindrome or not.
- Write a Python program to plot the function $y = x^2$ using the pyplot or matplotlib libraries.
- Create a graphical application that accepts user inputs, performs some operation on them, and then writes the output on the screen. For example, write a small calculator. Use the tkinter library.
- Open a webpage using the urllib library.
- Compute EMIs for a loan using the numpy or scipy libraries.
- Take a sample of 10 phishing e-mails and find the most common words.

2. **Data Management: SQL and web-server**

- Find the min, max, sum, and average of the marks in a student marks table.
- Find the total number of customers from each country in the table (customerID, customerName, country) using group by.
- Write a SQL query to order the (studentID, marks) table in descending order of the marks.
- Integrate SQL with Python by importing MYSQLdb.
- Write a Django based web server to parse a user request (POST), and write it to a CSV file.

6.4.9 Project

The aim of the class project is to create something that is tangible and useful. This should be done in groups of 2 to 3 students, and should be started by students at least 6 months before the submission deadline. The aim here is to find a real world problem that is worthwhile to solve. Students are encouraged to visit local businesses and ask them about the problems that they are facing. For example, if a business is finding it hard to create invoices for filing GST claims, then students can do a project that takes the raw data (list of transactions), groups the transactions by category, accounts for the GST tax rates, and creates invoices in the appropriate format. Students can be extremely creative here. They can use a wide variety of Python libraries to create user friendly applications such as games, software for their school, software for their disabled fellow students, and mobile applications, Of course to do some of this projects, some additional learning is required; this should be encouraged. Students should know how to teach themselves.

If three people work on a project for 6 months, at least 500 lines of code is expected. The committee has also been made aware about the degree of plagiarism in such projects. Teachers should take a very strict look at this situation, and take very strict disciplinary action against students who are cheating on lab assignments, or projects, or using pirated software to do the same. Everything that is proposed can be achieved using absolutely free, and legitimate open source software.

6.5 IP - 1(Class 11)

6.5.1 Prerequisites

Since a lot of students join CBSE schools from schools run by a state board, we are not assuming any pre-requisites for this course other than basic mathematical skills. However, it will be helpful if the student has a basic knowledge of Computer Applications 1 and 2.

6.5.2 Learning Outcomes

1. Basic computational thinking. Learn how to reason with variables, state transitions, conditionals, and iteration.
2. Notion of data types, and higher order data structures such as lists, and dictionaries.
3. Concepts of data handling, data manipulation, and the structure of simple SQL queries.
4. Cybersafety.

6.5.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Programming and Computational Thinking	30
2.	Data Handling - 1	20
3.	Data Management - 1	10
4.	Society, Law and Ethics - 1	10
5.	Practicals	30
	Total	100

6.5.4 Unit 1: Programming and Computational Thinking (PCT-1) (70 Theory + 60 Practical)

- Basic computer organisation: describe a computer system and mobile system, CPU, memory, hard disk, I/O, battery, power, transition from a calculator to a computer
- Familiarization with the basics of Python programming: a simple "hello world" program, process of writing a program, running it, and print statements; simple data-types: integer, float, string
- Introduce the notion of a variable, and methods to manipulate it (concept of l-value and r-value even if not taught explicitly)
- Knowledge of data types and operators: accepting input from the console, assignment statement, expressions, operators and their precedence.
- Conditional statements: if, if-else, if-elif-else; simple programs: e.g: absolute value, sort 3 numbers, divisibility.
- Notion of iterative computation and control flow: for, while, flowcharts, decision trees and pseudo code; write a lot of programs: interest calculation, EMI, tax calculation (examples from GST), standard deviation, correlation
- Lists and dictionary: finding the maximum, minimum, mean; linear search on a list of numbers, and counting the frequency of elements in a list using a dictionary.
- Text handling: compare, concat, and substring operations.
- Introduction to Python modules: creating and importing.

6.5.5 Unit 2: Data Handling (DH-1) (30 Theory + 20 Practical)

1. Introduction to Python Pandas
    - Introduction to data structures in Pandas: Series, and DataFrame
    - Operations on a Series: head, tail, vector operations
    - DataFrame operations: create, display, iteration, select column, add column, delete column
    - Binary operations in a DataFrame: add, sub, mul, div, radd, rsub
    - Matching and broadcasting operations
    - Missing data and filling values.
    - Comparisons, Boolean reductions, comparing Series, and combining DataFrames.
2. Transfer data between CSV files/SQL databases, and DataFrame objects.

6.5.6 Unit 3: Data Management (DM-1) (30 Theory + 20 Practical)

- Relational databases: idea of a database and the need for it, relations, keys, primary key, foreign key;
- Use SQL commands to create a table, keys, foreign keys; insert/delete an entry, delete a table.
- Basic SQL: select, project, and join; indexes, and a lot of in-class practice.

6.5.7 Unit 4: Society, Law and Ethics (SLE-1) – Cybersafety (10 Theory)

- Cybersafety: safely browsing the web, identity protection, confidentiality, social networks, cyber trolls and bullying
- Appropriate usage of social networks: spread of rumors, and common social networking sites (Twitter, LinkedIn, Facebook) and specific usage rules.
- Safely accessing web sites: adware, malware, viruses, Trojans
- Safely communicating data: secure connections, eavesdropping, phishing and identity verification.

6.5.8 Practical

S.No	Lab Task	Marks
1.	Programming in Python	10
2.	Data Management	10
3.	Data Handling	10
	Total	30

1. **Programming in Python:** At least the following Python concepts should be covered in the lab sessions: expressions, conditionals, loops, list, dictionary, and strings. The following are some representative lab assignments.
    - Find the largest and smallest numbers in a list.
    - Find the third largest number in a list.
    - Find the sum of squares of the first 100 natural numbers.
    - Find whether a string is a palindrome or not.

- Given two integers x and n, compute x^n.
- Compute the greatest common divisor and the least common multiple of two integers.
- Test if a number is equal to the sum of the cubes of its digits. Find the smallest and largest such numbers in the range of 100 to 1000.

2. **Data Management: SQL Commands**
   Refer to Section 6.3.8

3. **Data Handling:** The following are some representative lab assignments.
   - Subtract the mean of a row from each element of the row in a DataFrame.
   - Filter out rows based on different criteria such as redundant rows (same data as the row above or below).
   - Find the sum of each column, or find the column with the lowest mean.
   - Locate the 3 largest values in a data frame.
   - Replace all negative values in a data frame with a 0.

6.6 IP - 2 (Class 12)

6.6.1 Prerequisites

IP - 1

6.6.2 Learning Outcomes

1. Understand aggregation operations, descriptive statistics, and re-indexing columns in a DataFrame.
2. Apply functions row-wise and element-wise on a DataFrame.
3. Understand basic software engineering: models, activities, business use-case diagrams, and version control systems.
4. Connect a Python program with a SQL database, and learn aggregation functions in SQL.
5. Have a clear understanding of cyberethics and cybercrime. Understand the value of technology in societies, gender and disability issues, and the technology behind biometric ids.

6.6.3 Distribution of Marks

Unit No.	Unit Name	Marks
1.	Data Handling - 2	30
2.	Basic Software Engineering	15
3.	Data Management - 2	15
4.	Society, Law and Ethics - 2	10
	Total	70

6.6.4 Unit 1 : Data Handling (DH-2) (80 Theory + 70 Practical)

1. Python Pandas
   - Advanced operations on DataFrames: pivoting, sorting, and aggregation
   - Descriptive statistics: min, max, mode, mean, count, sum, median, quartile, var
   - Create a histogram, and quantiles.
   - Function application: pipe, apply, aggregation (groupby), transform, and applymap.
   - Reindexing, and altering labels.
2. Numpy
   - 1D array, 2D array
   - Arrays: slices, joins, and subsets
   - Arithmetic operations on 2D arrays
   - Covariance, correlation and linear regression
3. Plotting with Pyplot
   - Plot bar graphs, histograms, frequency polygons, box plots, and scatter plots.

6.6.5 Unit 2: Basic Software Engineering (BSE) (25 Theory + 10 Practical)

- Introduction to software engineering
- Software Processes: waterfall model, evolutionary model, and component based model
- Delivery models: incremental delivery, spiral delivery
- Process activities: specification, design/implementation, validation, evolution
- Agile methods: pair programming, and Scrum
- Business use-case diagrams
- Practical aspects: Version control system (GIT), and do case studies of software systems and build use-case diagrams

6.6.6 Unit 3: Data Management (DM-2) (20 Theory + 20 Practical)

- Write a minimal Django based web application that parses a GET and POST request, and writes the fields to a file – flat file and CSV file.
- Interface Python with an SQL database
- SQL commands: aggregation functions, having, group by, order by.

6.6.7 Unit 4: Society, Law and Ethics (SLE-2) (15 Theory)

- Intellectual property rights, plagiarism, digital rights management, and licensing (Creative Commons, GPL and Apache), open source, open data, privacy.
- Privacy laws, fraud; cyber crime – phishing, illegal downloads, child pornography, scams; cyber forensics, IT Act, 2000.
- Technology and society: understanding of societal issues and cultural changes induced by technology.
- E-waste management: proper disposal of used electronic gadgets.
- Identity theft, unique ids, and biometrics.
- Gender and disability issues while teaching and using computers.
- Role of new media in society: online campaigns, crowdsourcing, smart mobs
- Issues with the internet: internet as an echo chamber, net neutrality, internet addiction
- Case studies – Arab Spring, Wikileaks, Bitcoin

6.6.8 Practical

S.No	Lab Task	Marks
1.	Data Management (SQL + web-server)	5
2.	Data Handling	10
3.	Project Work	15
	Total	30

1. **Data Management: SQL+web-server**
   Refer to Section 6.4.8
2. **Data handling using Python libraries** The following are some representative questions.
   - Use map functions to convert all negative numbers in a DataFrame to mean of all the numbers.
   - Consider a DataFrame, where each row contains the item category, item name, and expenditure. Group the rows by the category, and print the total expenditure per category.
   - Given a Series, print all the elements that are above the 75^{th} percentile.
   - Given a days worth of stock market data, aggregate it. Print the highest, lowest, and closing prices of each stock.
   - Given sample data, plot a linear regression line.
   - Take data from government web sites, aggregate and summarize it. Then plot it using different plotting functions of the PyPlot library.
3. Basic Software Engineering
   - Business use-case diagrams for an airline ticket booking system, train reservation system, stock exchange
   - Collaboratively write a program and manage the code with a version control system (GIT)

6.6.9 Project

Regarding the project, refer to Section 6.4.9.

7 Methods to Operationalise the Curriculum

Let us first consider the two main recommendations: create a consultative process, and discontinue the older curriculum in a time bound manner.

The first recommendation of the committee is to make this document public, and ask for public comments. There are several advantages of this process. First is that it will create an environment that is based on trust and transparency, and also the stakeholders – students, teachers, and parents – will be able to understand the logic and rationale behind designing the new curriculum. They will also understand why a given topic is being removed, and other newer topics are being introduced. The committee is of the view that any process, which is opaque is bound to generate suspicion and controversy. In comparison, an open, systematic, and scientific process sets in forth a robust mechanism to effect curricular interventions. Moreover, after interactions with parents, the committee members noted that the school teacher is typically the first point of contact. If the school teacher is misinformed, then parents and students also get misinformed. It is thus necessary to have a document in the public domain, such that parents can consult it to get an alternative perspective. Secondly, parents in large cities can always ask friends and colleagues who are computer aware about the relative strengths of the new curriculum. However, in remote areas of the country, where such support is not available, parents will really benefit by having access to a written document of this kind.

The second recommendation is to make the new curriculum mandatory at the level of class 9 in 2018. For class 11 the recommendation is to offer both the curricula – new and old – in 2018, and then discontinue the older curriculum in 2019.

In other words, for class 11, 2018 entry students will have a choice between the new and old curriculum. However, in the old curriculum the MWT (Multimedia and Web Technologies) course will not be offered. Since some amount of additional learning is involved, it is best to give teachers more time (if they need). In the view of the committee we can have one year of overlap (for 2018). From 2019 onwards the new curriculum should be made mandatory in schools for class 9 and 11, and the old curriculum should be discontinued. This will give teachers, and creators of books and support material roughly 15 months for preparation. The committee feels that this is more than enough. For class 11, having an overlap of more than one year is not advisable. It defeats the purpose of having a novel and modern curriculum. A clarification is due here. Note that the new curriculum is applicable to only those batches that are **entering** either class 9 or class 11. It is mandatory for class 9 in 2018, and is optional for class 11 in 2018. It is compulsory for both classes 9 and 11 in 2019. This further means that any student who has had C++ (old curriculum) in class 11, will write the C++ paper in the board exam. He/she will not be a part of the new curriculum. In other words, no student will be asked to migrate to the new curriculum mid-way.

Let us now cover other issues regarding learning material, teacher training, methods to teach children with special needs, the examination pattern, and a feedback mechanism.

7.1 Learning Material

A common complaint that the committee heard in school visits is that NCERT does not have a textbook in this area. Students often have to refer to books written by various private publishers – some of which are of questionable quality. **Having a textbook is an absolute necessity.** It is the only reliable source of information for students. The rest of the sources such as websites, blogs, videos, online courses, and unverified books by other publishers, can at best be viewed as references. However, having one well written book is every child's right. The committee thus recommends that immediate efforts should be made for preparing three books for the Computer Applications, Computer Science, and IP courses respectively. The books will establish an appropriate baseline, and give proper guidance to students and teachers. In addition, immediate efforts to write lab manuals, and teacher training guides should also be made.

There should also be a significant e-learning component, given that it is very easy to produce such content now. The committee is of the view that Professors in IITs, NITs, and IIITs should be approached to create content for school students, which they can freely access over YouTube or government portals such as Swayam. This can be similar to the IIT-PAL initiative, which aims at creating content for IIT JEE aspirants. Teachers can also play this content in classes, and students can also refer to this content for learning at home.

The committee also noted in their interaction with parents that a lot of children are taking private tuitions, which are fairly extensive. If they have access to good quality content that is available free of cost, and also have a large repository of problems and solutions, then the need for expensive tuitions might reduce. Children can learn in the comfort of their homes.

7.2 Teacher Training

It is very important to train teachers such that they can deliver the desired learning outcomes. This is a fairly extensive exercise and requires multiple approaches.

The first is teacher training workshops. They can be conducted physically by professors from universities and engineering colleges. They can also be conducted virtually (on Webex), or by using MOOCs technologies. The aim should be to reach as many teachers as possible, and to also create enough content such that teachers who will be hired in the future can also be trained. A lot of these workshops can be videotaped and archived on MOOCs platforms such that they are available at a later date.

Note that training teachers is a common concern worldwide [Hubwieser et al., 2015b]. Here are some of the models that have worked in different countries:

1. In Germany, there is a mechanism of having intern teachers, where a junior teacher can work along with a senior teacher and learn the required skills.

2. France offers extensive training courses to teachers, and provides many options for certification during their service.

3. In New Zealand, one academic at a local university is assigned to advise schools in the neighbourhood. The academic arranges training sessions, and also mentors teachers and senior students.

4. Israel has mandatory in-service courses for CS teachers. They have a dedicated centre to conduct these workshops, train and assess teachers. This centre is responsible for continuous quality improvement.

5. In South Korea, for a teacher to maintain her teacher's license, she needs to attend a 60 hour training programme each year during holidays.

6. Russia has a wide network of both formal and informal teacher training centres. These centres are responsible for maintaining the pedagogical quality in schools.

7. In the UK they have created a network of centres for monitoring and improving CS teaching in schools. This centre has ample funding from government sources, and the private sector. It is meant to be a self sustaining model, where it can over time draw donations from private individuals, and companies, and subsequently expand its reach.

As we see, there is a spectrum of solutions. Israel and South Korea have extremely well organized systems, in comparison, New Zealand, Russia, and UK have opted for decentralized systems, which are also working very well. The committee advocates a middle path, where we can have regular workshops, which teachers can attend virtually (via Swayam or YouTube). There should be a process of regular certification, and in tandem a process of active engagement should be pursued with local universities.

7.3 Textbook Modifications for Children with Special Needs

Based on the wise counsel of experts, the committee has the following recommendations for creating learning material for children with special needs:

1. For every concept, have examples that do not use visual concepts such as appearance, and colour. It is okay to have examples centred around visual themes. However, provide alternatives to blind students. Basically, have additional examples.

2. Similarly for Autistic children, always add alternative examples that do not use social themes, or are not in a social setting.

3. Write the textbooks carefully, where you avoid a large *inductive leap*. An inductive leap is defined as a stretch of imagination that draws a reasonable inference from the available information (`https://www.virtualsalt.com/think/induct.htm`). Introduce concepts gradually, and always increase the complexity in small micro-steps. For example, do not immediately jump to *for* loops after introducing *if* statements. First introduce the concept of *goto* (albeit hypothetically), then write if-then-else logic with *goto* statements, then simulate a loop with 2 iterations, and then gradually increase the number of iterations. This will ensure that students who have issues with generalization, and abstract thinking, will be able to pick up the concepts by following the book. It is important to remember that at every step reduce the height of the inductive leap.

4. The knowledge base should be gradually expanded. Start with what children already know, and then gradually introduce concepts by taking baby steps. Note that the process of learning is two steps forward, and one step backward. Hence, write the book in a way that ensures that students can periodically revise what they have learnt. Students should be given enough opportunities to recapitulate.

5. Introduce motivational examples, and important way points (summary up till now sections) throughout the book.

6. Have graphical diagrams of a very high quality. In addition, explain the diagrams very well such that even blind students can get a gist of what is being explained. Consider using some of the new Braille diagram technology that is being developed by many universities in the world.

7. For students who cannot afford many program-run-debug cycles because of typing for them is a challenge, teach them how to reduce the number of errors by having more assert statements, and checking for invariants.

7.4 Examination Pattern

The committee looked at most of the past CBSE exam papers. It is of the view that the difficulty levels have to be increased, if better learning outcomes are to be ensured. There is a need to have a mix of easy and hard questions. At least one or two questions should be fairly difficult. Note that the job of an exam is not just to give a score to passing out students, it is also to bring in excellence into the system. This can only be done when students engage in serious critical thinking, and the exam is a very potent mechanism to ensure the same.

Based on analyses of histograms of marks in CBSE board exams, the committee has the following recommendations in an exam paper:

1. 40% of the questions are easy. These are short answer or multiple choice questions, where we are testing if a student has a very basic understanding of a concept.

2. 35% of the questions are of moderate difficulty. These questions require some additional thinking, where the inductive leap is moderate. In this category, there should be questions regarding finding errors in 5-10 line programs, and writing 3-4 line programs with loops. Questions in computer systems, networking, and SQL will have an equivalent complexity.

3. 20% of the questions are hard. These questions need a student to use all the features of the programming language, should possibly combine concepts from different areas, and require some degree of abstract thought. Every question in this category should require several minutes of thinking.

4. 5% of the questions are very hard. They are meant for the top 1% of the students. These questions should be significantly complicated, and should require a thorough knowledge of computational thinking. It should be based on programming, and should be at the level of a first year course in Engineering in a premier institute.

7.5 Feedback Mechanism

A curriculum design process is a continuous process. There should be a mechanism to collect feedback from a subset of schools spanning social strata. This information will be used to fine tune the curriculum. Furthermore, there should be a mechanism – with adequate legal permissions and safeguards – to find out how children with special needs are coping with the curriculum. Furthermore, the recommendation is that the board should create a mechanism – again with all legal and ethical safeguards – such that top researchers can develop affordable assistive technologies for children with special needs. To do this they need access to appropriate and relevant data, and secondly there should be pilot programs where students are given assistive devices for free. They can then provide their valuable feedback such that these devices can be further improved.

8 Conclusion and Afterthoughts

The committee worked very diligently to design a futuristic curriculum for the next generation. As far as possible an effort was made to make the process open, transparent, consultative, and participatory in nature. A lot of experts from different domains were consulted and an effort was made to get the best possible talent on board to design this curriculum. However, no endeavour is perfect. There is always a scope for improvement. The hope is that this document will serve as a baseline for subsequent curricular interventions, and we shall finally have a curriculum that will create some of the best computer literate individuals in the world.

Committee Chairs: Prof. Smruti R. Sarangi and Prof. Vikram Goyal

References

[Amelia McNamara, 2014] Amelia McNamara (2014). Introduction to data science for high school students. `http://datascience.la/introduction-to-data-science-for-high-school-students/`. Accessed on 28th November, 2017.

[Australian Curriculum Assessment and Reporting Authority, 2017] Australian Curriculum Assessment and Reporting Authority (2017). Australian curriculum. `https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/digital-technologies/`. Accessed on 22nd November, 2017.

[Baron et al., 2014] Baron, G.-L., Drot-Delange, B., Grandbastien, M., and Tort, F. (2014). Computer science education in french secondary schools: Historical and didactical perspectives. *Trans. Comput. Educ.*, 14(2).

[Bell et al., 2010] Bell, T., Andreae, P., and Lambert, L. (2010). Computer science in new zealand high schools. In *Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103*, pages 15–22. Australian Computer Society, Inc.

[Bell et al., 2014] Bell, T., Andreae, P., and Robins, A. (2014). A case study of the introduction of computer science in nz schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):10.

[Bocconi et al., 2016] Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., et al. (2016). Developing computational thinking in compulsory education-implications for policy and practice. Technical report, Joint Research Centre (Seville site).

[Brown et al., 2014] Brown, N. C., Sentance, S., Crick, T., and Humphreys, S. (2014). Restart: The resurgence of computer science in uk schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):9.

[Committee, 2016] Committee, K.-. C. S. F. S. (2016). K-12 computer science framework. Technical report, New York, NY, USA.

[Davies et al., 2010] Davies, T. C., Mudge, S., Ameratunga, S., and Stott, N. S. (2010). Enabling self-directed computer use for individuals with cerebral palsy: a systematic review of assistive devices and technologies. *Developmental Medicine & Child Neurology*, 52(6):510–516.

[Ezer and Harel, 1999] Ezer, J. G. and Harel, D. (1999). Curriculum and course syllabi for a high-school program in computer science. *Computer Science Education*, 9(2):114–147.

[Gardiner, 2014] Gardiner, B. (2014). Adding coding to the curriculum. `https://www.nytimes.com/2014/03/24/world/europe/adding-coding-to-the-curriculum.html`.

[Gould et al.,] Gould, R., Machado, S., Ong, C., Johnson, T., Molyneux, J., Nolen, S., Tangmunarunkit, H., Trusela, L., and Zanontian, L. Teaching data science to secondary students–the mobilize introduction to data science curriculum.

[Guo, 2015] Guo, P. (2015). Python is now the most popular introductory teaching language at top us universities (2014). *Communications in ACM, Blogs*.

[Heintz et al., 2016] Heintz, F., Mannila, L., and Färnqvist, T. (2016). A review of models for introducing computational thinking, computer science and computing in k-12 education. In *Frontiers in Education Conference (FIE), 2016 IEEE*, pages 1–9. IEEE.

[Hubwieser, 2012] Hubwieser, P. (2012). Computer science education in secondary schools–the introduction of a new compulsory subject. *ACM Transactions on Computing Education (TOCE)*, 12(4):16.

[Hubwieser et al., 2015a] Hubwieser, P., Armoni, M., and Giannakos, M. N. (2015a). How to implement rigorous computer science education in k-12 schools? some answers and many questions. *ACM Transactions on Computing Education (TOCE)*, 15(2):5.

[Hubwieser et al., 2014] Hubwieser, P., Armoni, M., Giannakos, M. N., and Mittermeir, R. T. (2014). Perspectives and visions of computer science education in primary and secondary (k-12) schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):7.

[Hubwieser et al., 2015b] Hubwieser, P., Giannakos, M. N., Berges, M., Brinda, T., Diethelm, I., Magenheim, J., Pal, Y., Jackova, J., and Jasute, E. (2015b). A global snapshot of computer science education in k-12 schools. In *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 65–83. ACM.

[Hubwieser et al., 2016] Hubwieser, P., Hubwieser, E., and Graswald, D. (2016). How to attract the girls: Gender-specific performance and motivation in the bebras challenge. In *Informatics in Schools: Improvement of Informatics Knowledge and Perception - 9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2016, Münster, Germany, October 13-15, 2016, Proceedings*, pages 40–52.

[Hugo Moreno, 2017] Hugo Moreno (2017). Data analytics is no longer a nice option – it's the core of the enterprise. `https://www.forbes.com/sites/forbesinsights/2017/06/12/data-analytics-is-no-longer-a-nice-option-its-the-core-of-the-enterprise/#1919d5e177ec`. Accessed on 22nd November, 2017.

[Jones, 2013] Jones, S. P. (2013). Computing at school in the uk.

[Jones et al., 2011] Jones, S. P., Bell, T., Cutts, Q., Iyer, S., Schulte, C., Vahrenhold, J., and Han, B. (2011). Computing at school. *International comparisons. Retrieved May*, 7.

[Khenner and Semakin, 2014] Khenner, E. and Semakin, I. (2014). School subject informatics (computer science) in russia: Educational relevant areas. *ACM Transactions on Computing Education (TOCE)*, 14(2):14.

[Man and Wong, 2007] Man, D. W. and Wong, M.-S. L. (2007). Evaluation of computer-access solutions for students with quadriplegic athetoid cerebral palsy. *American Journal of Occupational Therapy*, 61(3):355–364.

[PTI, 2017] PTI (2017). 95 `http://www.thehindubusinessline.com/info-tech/95-engineers-in-india-unfit-for-software-development-jobs-study/article9652211.ece`.

[Raman et al., 2015] Raman, R., Venkatasubramanian, S., Achuthan, K., and Nedungadi, P. (2015). Computer science (cs) education in indian schools: Situation analysis using darmstadt model. *ACM Transactions on Computing Education (TOCE)*, 15(2):7.

[Rica Bhattacharya, 2017] Rica Bhattacharya (2017). Mncs embrace autistic people to improve diversity. `https://economictimes.indiatimes.com/news/company/corporate-trends/mncs-embrace-autistic-people-to-improve-diversity/articleshow/61788800.cms`. Accessed on 25th November, 2017.

[Rivers et al., 2016] Rivers, K., Harpstead, E., and Koedinger, K. R. (2016). Learning curve analysis for programming: Which concepts do students struggle with? In *ICER*, pages 143–151.

[Shriram Krishnamurthi and Emmanuel Schanzer, 2017] Shriram Krishnamurthi and Emmanuel Schanzer (2017). Bootstrap's data science course for middle- and high-school students. `https://www.techatbloomberg.com/blog/bootstraps-data-science-course-for-middle-and-high-school-students/`. Accessed on 22nd November, 2017.

[Sinha et al., 2014] Sinha, P., Kjelgaard, M. M., Gandhi, T. K., Tsourides, K., Cardinaux, A. L., Pantazis, D., Diamond, S. P., and Held, R. M. (2014). Autism as a disorder of prediction. *Proceedings of the National Academy of Sciences*, 111(42):15220–15225.

[Taylor, 2017] Taylor, M. (2017). Computer programming with early elementary students with and without intellectual disabilities.

[Tucker, 2003] Tucker, A. (2003). A model curriculum for k–12 computer science: Final report of the acm k–12 task force curriculum committee. Technical report, New York, NY, USA. ACM Order No.: 104043.

[Wing, 2014] Wing, J. (2014). Computational thinking benefits society. *40th Anniversary Blog of Social Issues in Computing*, 2014.