A computational thinking requirement for MIT undergraduates

Report of the working group on computational thinking January 2017

Summary of findings:

During the spring of 2016, Chair of the Faculty Krishna Rajagopal and Dean for Undergraduate Education Denny Freeman assembled a group of faculty, representing all five schools of the Institute, to conduct an in-depth study of the role of "algorithmic reasoning/computational thinking" in the context of the education of MIT undergraduates. The group was asked to consider a set of questions relating to this topic, including whether formal exposure to algorithmic/computational thinking should be required of all MIT undergraduate students.

A summary of the key findings of the working group includes:

- Computational thinking should play an explicit role in the formal education of all undergraduate students at MIT. Computational thinking provides a distinct type of rigorous thought of important intellectual value; it requires and develops important modes of communication; it acknowledges the need to understand the transformational impact of computation in other disciplines; and it creates opportunities and access for our students and graduates.
- In response to varied community interpretations of the topic, the working group developed a notion of computational thinking and algorithmic reasoning, defining them both in and of themselves and in relation to other modes of thought. This notion includes the assertion that computational thinking is broader than a proficiency in computer programming, although programming languages provide a particularly useful framework for understanding the fundamentals and applications of computational thinking.
- While a significant portion of the student body currently takes a relevant course in computation, coverage is not universal. The working group believes that just as every student learns critical thinking and inductive and deductive reasoning as pathways to analysis, understanding and discovery through their humanities, arts and social science subjects and through the current science General Institute Requirements, so too should every student learn computational thinking. Thus, the working group recommends that all undergraduates be required to take at least one subject offering in computation.
- Computational thinking involves more than the skill of computer programming or the ability to use computer tools; it includes fundamental modes of reasoning about the rendering of physical or social systems in a

manner that enables computational experiments to complement physical or social ones. Formal exposure to these concepts is important for all students.

- The working group recommends that the Institute proceed with a consideration of mechanisms by which a computation requirement could be instituted for all undergraduate students, while addressing the impact adding an additional degree requirement or substituting a current requirement would have on student load and while addressing the need to connect computational thinking to domain-specific contexts across different intellectual disciplines. The working group also recommends that this consideration examine the impact of potential changes in requirements on ABET accreditation of engineering degrees.
- The working group report outlines several ways in which a requirement in computational thinking might be implemented, and explores the advantages and challenges of each approach. Options for incorporating a computational requirement must carefully consider the tight constraints imposed on students by the current Institute-wide requirements and by additional departmental degree requirements, and should avoid adding a significant burden on our students.

Background:

During the spring of 2016, Chair of the Faculty Krishna Rajagopal and Dean for Undergraduate Education Denny Freeman assembled a group of faculty, representing all five schools of the Institute, to conduct an in-depth study of the meaning of the phrases "algorithmic reasoning" and "computational thinking" in the context of the education of MIT's undergraduates across all five schools. Their charge to this group included the following questions:

1) How do faculty, students and alumni in different fields of endeavor, across the full breadth represented by MIT's five schools, use computational thinking?

2) What, if any, is the common intellectual framework that people across MIT employ when they speak of computational thinking and algorithmic reasoning?

3) To what extent are algorithmic reasoning and computational thinking already being taught?

4) Should we acknowledge algorithmic and computational thinking as an explicit expectation of all our graduates?

5) If yes, what are the key elements of algorithmic and computational thinking and what are the associated learning objectives and measurable outcomes for knowledge, skills and attitudes?

6) If yes, does it matter when during their careers at MIT our students are exposed to computational thinking and algorithmic reasoning?

7) What are our peer institutions doing?

For all of these questions, the working group was charged to consider the diversity of meanings, of modes of knowledge, and of learning objectives across MIT's schools and the range of endeavors of our students and alumni.

This report summarizes the findings of the group in response to these questions. Based on the group's findings, the report articulates some potential actions and changes in requirements, any of which would advance the goal of improving the computational thinking skills of MIT undergraduates. The report does not specify any particular change in policy nor the specific details necessary to implement any of these improvements, since these recommendations were not part of the charge.

Process:

The working group engaged in a series of activities to solicit input from across the Institute, including the following:

- The Chair of the Faculty and the Dean for Undergraduate Education sent an email to every faculty member soliciting input. The working group discussed the 17 responses received by email, as well as additional comments and suggestions provided in person by others.
- The Chair and Dean sent a similar message to the student body of MIT. Though we expect that student interest in the topic is high, only 2 responses were received and discussed, perhaps because of the timing of the working group's activities.
- Members of the working group contacted every academic department head, and either engaged directly with the department head or with a designated representative, soliciting input on the questions posed in the charge to the group.
- The working group solicited data from the Registrar on recent enrolments of undergraduate students in classes that teach and cultivate computational thinking.
- The working group examined the curricula and requirements of peer institutions.
- Meeting extensively over the late spring and summer terms, the working group debated and discussed issues and options related to the charge.
- The working group issued a draft report early in the fall term of 2016, and solicited comments on this report from the entire MIT community. Comments were received from individual faculty, students and alumni. Comments were also received from the Committee on the Undergraduate Program, and from the Academic Advisory Council of the Undergraduate Association. All comments were then discussed by the working group and appropriate revisions to the report based on that discussion were incorporated into the final report.

What is "computational/algorithmic thinking" and is it relevant to the education of every MIT undergraduate student?

The working group spent considerable time discussing the concepts of computational thinking and algorithmic reasoning. The term "computational thinking" dates at least back to Seymour Papert in 1980, although a commonly accepted formal definition of the term is still under debate. One common definition of computational thinking, attributed to Jeannette Wing, is:

> *the thought processes involved in formulating a problem and expressing its solution(s) in such a way that an information processor – human or machine – can effectively carry out that solution*.

Such a definition clearly overlaps with several other formalized modes of thinking (such as mathematical or logical); however, the focus on a detailed, ordered sequence of operations that can be executed in a disciplined manner (i.e., algorithmic design) by an information processor distinguishes this approach from

others. This definition also distinguishes computational thinking from computer literacy (the ability to use computational tools as black box abstractions) or the knowledge of a specific programming language.

While the terms "computational thinking" or "algorithmic reasoning" may have different connotations in different disciplines, there is a clear sense across the Institute that there is a common experience of using computation as a mode of thought, and that this is more than acquisition of programming skills. This perspective has been articulated in the past, for example, in the 2010 National Research Council's report on Computational Thinking:

> *As the use of computational devices has become widespread, there is a need to understand the scope and impact of what is sometimes called the Information Revolution or the Age of Digital Information…[H]owever, most efforts have not focused on fundamental concepts.*

The report continues by listing three common approaches (computer literacy, particular programming languages, and programming applications) before distinguishing computational thinking as independent from any of them:

> *But in the view of many computer scientists, these three major approaches—although useful and arguably important—should not be confused with learning to think computationally. In this view, computational thinking is a fundamental analytical skill that everyone, not just computer scientists, can use to help solve problems, design systems, and understand human behavior. As such, they believe that computational thinking is comparable to the mathematical, linguistic, and logical reasoning that is taught to all children. This view mirrors the growing recognition that computational thinking (and not just computation) has begun to influence and shape thinking in many disciplines—Earth sciences, biology, and statistics, for example. Moreover, computational thinking is likely to benefit not only other scientists but also everyone else—bankers, stockbrokers, lawyers, car mechanics, salespeople, health care professionals, artists, and so on.*

The views expressed in that summary loudly echo what the working group found in its own investigations and discussions – that computational thinking involves more than the skill of computer programming or the ability to use computer tools such as spreadsheets or visualization programs, and that the importance of a grounding in computational thinking, especially for MIT undergraduates, is broader than specific disciplinary needs for computer programming skills.

While there may not be a universally accepted definition of computational thinking, while the particular instantiation of that idea may vary with the specifics of a discipline, and while there are clear ties between computational thinking and other forms of reasoning (mathematical, logical, linguistic), there is also a widely shared sense among the faculty across the Institute that an understanding of the

foundations and tools of computation – e.g., abstraction, modularity, recursion and iteration, divide-and-conquer, approximation and convergence – is a critical tool for any student.

The working group thus believes that computational thinking is more than just the skill of computer programming. It involves understanding how to model large scale systems using appropriate levels of abstraction and modularity, it utilizes mechanical descriptions of inference to analyze complex data collections, and it provides a computational complement to physical experiments on real-world problems. It is, however, hard to grasp or articulate concepts of computational thinking absent facility in a distinct and unambiguous language in which to describe them. And this requires the use of a computer programming language as a framework within which to explore computational concepts.

A few students suggested, in reaction to the draft report, that one could acquire facility in computational thinking without learning a specific programming language; instead focusing on theoretical descriptions and formal proofs. The working group does not support this position. As articulated above, knowing how to program a computer (as opposed to merely understanding the syntax of a computer language) is a requisite skill for computational thinking.

In addition, the ability to program a computer is an important life skill, for several reasons:

- Knowledge of computer programming (i.e., the ability to write code) enables a person to sift through vast amounts of information. It also permits a person to combine information from multiple sources in useful ways.
- While computational reasoning in the absence of programming would indeed demonstrate that an answer provided by the program *would* be correct, programming knowledge allows the correct answer to be determined, according with MIT's mission of providing solutions to tackle the world's great challenges.
- Computer programming skills are closely related to the skills needed to create large, complex models of social, physical, or biological systems.

The concepts of computational thinking, especially as a framework for rigorous reasoning about physical and social systems, can provide a basis for augmenting other intellectual frameworks. Many faculty members consider computational models to be important complements to theoretical and experimental models. Just as historically scientists and engineers used theoretical models to guide the design of experimental validations, and experimental observations to inform the creation of theoretical models, today engineers, natural scientists and social scientists also use computational models of physical or social systems to enhance understanding of observed or predicted phenomena. For these colleagues, the increasing reliance on using computational models to further their research (in basic science, in development of technology, or in modeling of social systems) requires a

fundamental understanding of computation – not just as a programming tool, but also as a foundational substrate on which to build and test models. Thus, they suggest that students should experience a subject that uses computation to understand physical and social systems, and that such a subject should include significant programming experience.

In this view, computation can become a framework in which to understand physical and social systems, to complement theoretical and experimental models with computational ones, and to use programming as a tool to ground that understanding of a system and to devise computational experiments to complement physical ones.

What elements are essential to computational thinking?

Given a perspective on computational thinking as *"the thought processes involved in formulating a problem and expressing its solution(s) in such a way that an information processor – human or machine – can effectively carry out that solution"*, one can then ask whether there are essential concepts that would form the basis for proficiency in computational thinking. While such a discussion may vary across disciplines, the working group found that the following basic concepts are widely accepted as key elements in computational thinking:

- Abstraction of processes: capturing common patterns of operation in an algorithmic description that can be generalized and applied to multiple instances of a problem. This abstraction includes the suppression of details of the procedure from users (whether humans or other computational processes), so that subsequent computational processes can utilize the behavior of an abstraction without regard for the specifics of how it produces results.
- Abstraction of data: capturing patterns of association within complex collections of data to structure it so that it can be appropriately/efficiently/accurately processed. This includes identification of associated elements of data, and abstracting out specific details of representation of data elements from the use of such data.
- Decomposition: reducing a computational task into a sequence of simpler tasks, together with mechanisms for integrating the results of that decomposition into the solution to the original problem.
- Modularity: understanding how to characterize the solution of a computational problem as a sequence of operations, each of which can be represented as a separate, abstract, computational problem; together with defined operations for integrating the results of each sub-operation into a solution for the original problem.
- Iteration and recursion: understanding fundamental approaches for successively reducing the solution to a complex problem into simpler versions of the same problem.

These concepts can be understood abstractly; however, the working group believes that such concepts are best understood within the specifics of a language of description – a programming language that provides the primitives and means of combination in which to describe and deploy these concepts on real problems.

As a consequence, the working group has used the following description as a framework in its deliberations:

> **On computational thinking:**
>
> *When our working group uses the term "computational thinking", we mean something more than learning the syntax of a computer language. We want students to develop skills and modes of thinking so that they can construct or recognize useful, well written algorithms, can implement them, and can use them to model physical, biological, or social systems.*
>
> *Even in this limited sense, any proposed requirement in computational thinking should not require that MIT students develop all of the many skills that are useful in this area. Rather, any proposed requirement should ask them to learn some of the fundamental skills, and to practice these skills by developing algorithms and writing computer programs. Such an experience should provide a solid foundation on which subsequent exploration of computation, in a wide variety of disciplines, can be undertaken.*

With that perspective, and based on the information gathered from across the Institute, the working group believes that computational thinking should play a role for students in all parts of the Institute. This role is based on the new intellectual mode of thought that computational thinking provides, rather than purely on the pragmatic advantages these tools might give at MIT or in professional life, although we acknowledge that often one must use a programming language as a framework within which to describe aspects of computational thinking. We further believe that there are several important reasons why **every** MIT undergraduate student should be articulate in the role of computation as a mode of thought and a means of communication:

1. **Computational thinking is a distinct type of rigorous thinking that is of intellectual value**. Computational thinking, similar to mathematical or logical thinking, combines structured reasoning with creative exploration of paths to reach a result. Computational thinking differs from mathematical thinking, however, in the emphasis it places on managing complexity, limiting resources, and effectiveness in modeling physical and social systems. It also differs in that it stresses imperative (or "how to" knowledge) rather than declarative (or "what is true" knowledge). For example, axiomatic statements about properties of square roots are different from algorithmic methods to compute specific square roots. Additionally, computational

thinking often involves making informed decisions about tradeoffs, such as between model detail and computational speed, and understanding how these tradeoffs may have consequences for the accuracy, or even the social and ethical implications, of results. In this way, computational thinking adopts a uniform set of principles and clearly established tests of consistency that provide a distinct intellectual framework for thinking about physical and social models.

2. **Computational thinking requires and develops important modes of communication**. Most people have had the experience where their belief that they understand how something works is shattered the moment they try to describe that system in writing or teach it clearly to others. Written and oral expression crystallizes vague thoughts into concrete ideas. Computationally rigorous thought, expressed in code, leads to a clarity of design that communicates the beliefs of the code's author to others. Computational thinking involves making explicit hierarchical and modular relationships; algorithms and code communicate these relationships to others. Articulation of ideas in a manner that is comprehensible to others requires precision, clarity, and logical rigor, and thus computational expression can complement natural language as a means of communicating some ideas effectively.
3. **Computers are transformational agents in the 21st century.** They are changing the way that work is carried out. They are changing almost every field of endeavor. They have enormous impacts on our personal lives – through novel social interactions as well as in disrupting industries and professions. It is important for every student to develop an understanding of computers and what they can do. Even in fields where not all students use programming in their coursework, students should be cognizant of the impact of computation on their field. In this way, a knowledge of paradigms of computing is as important as the knowledge of the paradigms of, for instance, biology: just as a student may not use biological concepts in their area of interest, they should be cognizant of questions, challenges, and opportunities that arise in biology (or in physics, chemistry, mathematics, or the humanities, arts, and social sciences).
4. **Computational training creates opportunities and access for our students and graduates**. There are many opportunities available only to those who understand computational thinking and computer programming. These opportunities are available both to our students while they are at MIT (e.g. UROPs in virtually any discipline at MIT) and in their future careers. By increasing exposure to computational thinking to include all members of the MIT undergraduate student body, access to such opportunities can be expanded to students from a broader range of backgrounds, experiences, and points of view. This both opens new career possibilities for students and helps to increase diversity in computation-related fields in industry. Further, as computation continues to permeate other disciplines, knowledge of and

ability to use computational tools will become even more essential to success in a wide range of fields. To ensure that our graduates are well positioned to assume leadership roles in a variety of disciplines and industries, it is critical that they have a depth of understanding of computational issues and methods.

Elements of computational thinking:

While the working group believes that every undergraduate should gain exposure to computational thinking, it is hesitant to create a list of topics in a form like a syllabus. Development of appropriate subjects in computation may require specific sensitivity to disciplinary needs, an investigation that is beyond the scope of the working group's charge. However, there is wide agreement on the usefulness for all students to develop a strong understanding of the following topics:

1. **The fundamental constructs of computer programming and their roles in abstraction.** These would include the use of loops for iteration, as well as the use of recursion for algorithm design and implementation. It would also include the use of basic data structures such as lists and arrays, and the use of object classes as an abstraction mechanism for encapsulating data and associated methods of manipulation and inference. It would include the use of procedure definition and specification as an abstraction method. And it would include the importance of testing and debugging in the creation of robust algorithms and implementations.
2. **Elements of design for computer programming.** These include the importance of modular design and the role of abstraction mechanisms as a means of suppressing detail and supporting system design. It would also include methods for creating programs that make them easier to share, understand, test, and debug.
3. **Developing skills in at least one modern programming language.** This would include fundamental skills in writing and reading code, debugging, and related topics. While it is theoretically possible to apply algorithmic thinking without using programming, the ability to express such thinking through programming and to understand how computation actually works in society and industry is valuable in itself.
4. **Understanding and extending basic classes of algorithms.** This includes general-purpose approaches for solving difficult problems. Some examples include: (i) greedy algorithms, (ii) divide and conquer algorithms, (iii) the use of randomization and stochastic sampling in computer algorithms, (iv) hill climbing (or neighborhood search) algorithms, (v) interval bisection method, and others.
5. **Modeling our physical and social worlds.** This would include the ability to create mathematical and computational models that aid in understanding our physical and social worlds. It would also include learning the merits as

well as the limits of this type of modeling. This would also include grounding in methods for presenting and understanding results of computational experiments, such as visualization, and statistical analysis of uncertainty.

In addition to these elements at the core of any fundamental computational thinking requirement, the working group feels strongly that students should receive instruction in extensions of basic computation elements, and disciplinary applications to place the core concepts within diverse contexts. Some possible examples of such extended topics might include:

1. **Application of computational modeling within domain-specific contexts.** For example, the use of search algorithms in analyzing genomic data; modeling the mechanical properties of objects and their physical interactions; predicting patterns of behavior in large scale systems (transportation, communication, energy, social); and others.

2. **Understanding the limits of computers.** Part of understanding what computers can do is also understanding what they can't do (or can't do yet). It is also useful to use one's own knowledge and common sense to aid in knowing when computer output is not making sense or is not realistic. (This is an extension of recognizing when a calculator result does not make sense.)
3. **Visualization and non-textual interaction.** While many results of computational experiments are captured by numerical values, often the interpretation of those results is best done through other means. Visualization of a statistical distribution of trials of a computational experiment is an important aspect of understanding computational results.
4. **Computational creativity.** The use of computation to engage and augment human creativity in new ways, enhancing expression in art and design through the creative application of algorithms and code.

Thus, the working group unanimously believes that all MIT undergraduate students should have some mastery of computational thinking, as they do with physical, mathematical, biological and chemical thinking, and as they do with the critical thinking embedded in the humanities, arts and social sciences.

Perception across the Institute:

Although not unanimous, the working group found very broad support across the Institute for a mechanism by which all MIT undergraduates would gain competence in computational thinking, and in using computation (especially through program design and implementation) as a complement to other forms of intellectual inquiry. This support came from faculty and students in all five schools, although there were some dissenting views (concerns are discussed below). Not surprisingly, there was

strong support throughout the School of Engineering; however, there was also support (though not uniform) within the other four schools. Colleagues from disciplines as varied as Economics, Brain and Cognitive Sciences, Physics, Biology, Music, and Architecture all expressed support for the role of computational thinking, although we note that there were also some groups that did not see the need for inclusion of computational thinking for students in their sub-discipline.

Current coverage:

Although there is no requirement for computation that currently applies to all undergraduate students, there is already wide coverage of computational concepts for many of our students:

- All 8 departments in the School of Engineering have an explicit degree requirement of a class (or a portion of a class in one case) on computation. While the specifics of the subjects (such as their underlying programming language and their particular focus on algorithm development) differ across departments, and not all may cover all elements of computational thinking, all subjects require students to treat computation not just as a tool but as an ability to translate a logical description of a method into an algorithmic process.
- Currently, one department in the School of Science (Brain and Cognitive Sciences), one additional degree program in the School of Science (Mathematics with Computer Science), and one newly created degree program within the Sloan School of Management (Business Analytics) also have an explicit degree requirement of a subject on computation. (Previously the Management Science degree also had a required subject on computation.)
- In addition to coverage through degree requirements, many students take elective classes with computational concepts. To provide a sense of the number of students already gaining experience in computational thinking, we considered seniors who graduated in the years 2012 to 2016. Of that group, 4017 out of 5429 received a primary degree (not counting double majors) in a department that currently has a computational requirement. This represents 74% of the total cohort. Of the remaining 1412 students, 646 (or 46%) took at least one of 6.0001/6.0002 or 6.01 or 1.000 or 1.00 or 2.086, even though not required for their degree program (these four courses are not the only options for gaining experience in computation, but represent the four largest such options). Thus, out of 5429 students, only 766 were not required to take a computation course and did not take one of the four large ones (there may be other courses they did take that could be considered to satisfy this constraint), meaning that at most 14% of the graduating seniors in this cohort did not see computational thinking as part of their MIT education.

One could argue (and some faculty members do) that since the majority of MIT undergraduates already take at least one class in computation, there is no need to

require it of all students. However, MIT educational requirements are also a statement to our community and to the world of what MIT believes to be of the utmost importance in its undergraduate education. The working group believes that computational thinking is of such importance as to merit being required. Moreover, there is value in MIT stating that **all** of its students learn computational thinking. In support of this, consider the statement in MIT's Bulletin about the Science/Math GIRs:

> *"MIT expects its graduates to have an understanding and appreciation of the basic concepts and methods of the physical and biological sciences. … They are an essential part of the background that MIT graduates bring to their roles and professionals and as broadly educated citizens in a world strongly influenced by science and technology."*

In the view of the working group, this statement applies equally strongly to an understanding and appreciation of the basic concepts and methods of computation.

Additional considerations:

The charge to the working group raised some additional questions, which are briefly addressed below.

i) **When in their career do we expect students to learn computational thinking?** There is a range of views on this question. However, the working group agreed that ideally students would gain exposure to computational thinking early in their student experience. This view is shared by many departments. The working group is cognizant of the concern of overloading the first year with expectations of subjects to be taken, especially since so many first year students undertake most of the Science GIR requirements in that year. The group also felt, however, that many departments will expect their students to have multiple experiences with computation – both introductory and discipline specific – and thus an early exposure to foundational concepts would enable a more detailed, discipline-centric, exploration as an upperclassman. The working group notes that one of the options (discussed below) might enable this opportunity, by encouraging a 6-unit introductory subject early in a student's career, followed by a discipline-centric follow-on subject later.

ii) **Are comparable requirements in place at our peer institutions?** The working group was not able to conduct an extensive examination of peer institutions. We note the following observations:

a. The only peer institution that we are aware has an institute-wide requirement of a subject in computation is Harvey Mudd College, which describes itself as a science, engineering and mathematics liberal arts college (all nine of its majors are in science, engineering or mathematics). We are not aware of any other peer institution that has

an institute-wide requirement of a subject in computational thinking. Many institutions have introduced introductory computation courses intended for "non-majors" and encourage their students to take such courses, but none of which we are aware require such participation.

b. Some Schools of Engineering at peer institutions do have a computation requirement, as do some peer institutions with separate Schools of Computation, or Schools of Information Science (e.g., Cornell, Georgia Tech, Carnegie Mellon). Interestingly, it appears that not all engineering departments at Stanford have a specific computation course requirement, whereas all engineering departments at UC Berkeley and all engineering departments at Cornell do have such an explicit course requirement.

In general, we note that while we are not aware of peer institutions with specific institute-wide computation requirements, we do observe that many such institutions are moving towards encouraging students, especially in science and engineering, to acquire computational skills (for example, currently the most popular course at Harvard is CS50 – introduction to computation). Given these trends, there is an opportunity for MIT to lead the way in formalizing the expansion of education in computational thinking to include the entire undergraduate student body.

Concerns raised by members of the community

Members of the community raised many thoughtful comments and questions. We discuss these below:

1. To the extent that there was resistance to a required computational thinking experience, this was generally centered on the impact of adding a requirement on top of other existing requirements, especially the current Science GIR subjects. The working group agrees that MIT students cannot simply add another graduation requirement without negatively impacting their work and lives. We believe that any implementation of a change in requirements must consider the impact of potential changes on other parts of the MIT education and the intellectual benefits of an educational experience. This is discussed in more detail below.
2. While many colleagues feel that computational thinking should be an integral part of every student's undergraduate experience at MIT, many also feel that a discussion of changes in requirements should consider the broader question of other potential additions or changes. More specifically, the working group heard suggestions that just as computation has become an essential element in virtually every intellectual discipline at MIT, so too has statistics, probability and reasoning under uncertainty. While a discussion of the role of statistical reasoning in our curriculum is beyond the scope of this working group's charge, the group acknowledges that there is merit in considering the role of such a mode of thought in the MIT undergraduate

educational experience. But the working group also believes, especially since such a large percentage of our undergraduates already take at least one computation class, the Institute should move forward in considering possible implementation mechanisms for adding a formal requirement to the curriculum.

3. As noted above, a large portion of recent graduating classes have taken a computation class, either because it is a departmental degree requirement, or because of a personal interest in the area. In light of this, some faculty and students have questioned the need to incorporate a specific requirement for all students. Why should we require that every student take a subject in computational thinking? Why not let individual departments determine what is best for their students? In principle, the same argument could be applied to current science General Institute Requirements. Since most students would presumably take basic courses in calculus, physics, and perhaps biology and chemistry, why require that all students do so?

   That topic has been debated at the Institute in the past, including as part of the Silbey report on the General Institute Requirements. The generally accepted rationale for requiring all students to have competency in mathematics and sciences is that the modes of thought associated with those disciplines are important to understanding the world and the challenges confronting it. Many of our undergraduates will not use the specific elements of an introductory subject in biology or chemistry in any of their department-specific requirements, but the Institute feels that knowledge of biological reasoning is important to the broader education of all of our students. The question is whether the Institute also believes that computational reasoning rises to the same level of importance – that every student should be able to understand computational approaches to physical, biological, or social challenges, that every student should understand the impact of computational approaches in addressing such challenges, and that every student should understand how computational agents are changing virtually every aspect of modern life. The working group unanimously believes this to be case – that computational thinking is as essential a tool for every student as is scientific thinking. While not every student or faculty member agrees, the response to the draft report and the feedback acquired by the working group strongly suggests that that a substantial majority of the community also supports this perspective.
4. A question was raised about how our alumni and alumnae perceive the need for or value of exposure to computational thinking in their career paths. Although unfortunately we do not have explicit data from recent alumni/alumnae surveys on the role of computation in their careers, we have anecdotal feedback that suggests a strong sense of the need for computation, across a wide range of professions. Individual responses to the draft report, as well discussions with alumni/alumnae in other settings strongly support the role of understanding computation as an important element in their careers.

5. A similar question was raised about the perspective of employers on the need for all MIT undergraduate students to have significant exposure to computational methods. While no explicit survey of employers has been conducted, a variety of indirect measures suggest that there is a strong interest in such exposure. These include the distribution of companies that regularly participate in the annual career fair – a distribution that includes not only a very large presence of information technology companies, but also a large array of other companies explicitly seeking graduates with backgrounds in other disciplines who have exposure to computational methods. Additionally, demand from pharmaceutical companies, banks, investment firms, biotechnology companies, medical device companies, consulting companies, and others, for students with both a disciplinary knowledge of the sector and a background in computation is very strong, as evidenced by the hiring patterns of these companies and their expressed interest in MIT graduates.
6. Although the role of computational thinking in the sciences, in management, and in the engineering disciplines is well accepted, some students have questioned the importance of computational thinking in the humanities, arts and social sciences. While the working group believes that the arguments put forward for why every student should understand computation apply to the humanities as well as to the natural sciences, the group also notes that there are many areas of the humanities, arts and social sciences with natural interactions with computational thinking: economics, finance, linguistics, and others. But more generally, the working group stresses that computational thinking is not just about computational tools and their impact on work in a discipline, it is also about the impact that computation has in the framing of questions within a discipline. The group notes that many areas of humanities, arts and social sciences are already being changed by the emergence of computation – as a means of social interaction, as a facilitator of new modes of communication, as a curation mechanism for digital and other material, and as an influence on the central elements of disciplines. Thus knowledge of computation is likely to be of import in all of our academic disciplines.

Potential options for satisfying a computation requirement

Although the working group was not explicitly charged with devising an implementation plan, we were asked to articulate possible options for meeting the need for a common computational experience, including discussion of the strengths and weaknesses of such options. We do so below.

However, the working group also notes that any discussion of adding a general requirement for our undergraduate students may be best implemented if it happened in the context of a larger discussion of the General Institute Requirements. While many faculty members view computation as a critical mode of thought, there are other topics that may merit similar consideration. In particular, members of the

working group and many faculty members have proposed examining the state of instruction and importance of statistical thinking and reasoning under uncertainty. Thus, the working group encourages the Institute to consider any discussion of possible inclusion of a computation requirement within the broader context of all Institute requirements.

Given that such a broad discussion may not occur immediately or even in the medium term, the working group will discuss options for balancing a new computational requirement with a reduction in other requirements (specifically the REST requirement) below.

The working group lists, but does not necessarily endorse, each of the following possible options for satisfying a computational thinking requirement, while noting that there may be other possibilities (for example, the group discussed the option of integrating computation as a module into existing GIR subjects, either directly or as an online module, but concluded that these options were not compatible with the amount of material to be covered and the advantages of having that material taught by a faculty member with a computational background). The three options that the committee discussed at length were:

1. **A single, Institute-wide, subject:**
    a. **Description:**
        i. The Institute would create a single subject (or revise an existing subject) that incorporates commonly accepted elements of computational thinking including programming skills.
    b. **Advantages:**
        i. The development of a subject focused on a computational experience for all students would ensure that key concepts are presented to all students in a similar manner.
        ii. As with several of the other Institute wide requirements, there would be a shared experience for all students.
        iii. This would allow a focus on computation as a mode of thought, separate from embedding of computation in domain-specific contexts.
    c. **Challenges:**
        i. Students have vast differences of prior experience in computational work, especially programming, potentially leaving some students discouraged and overwhelmed and some other students bored and disengaged.
        ii. Giving students the option of passing out of the subject would address the latter issue, but would also exacerbate differences in the "real" number of requirements to graduate that already exist in other GIRs (for instance, because of prior Calculus experience or credits from AP classes). The requirement would disproportionately fall on students from high schools with

lower academic standards who already face additional difficulties.

iii. A single subject could not cover all the basic and extended topics discussed above, and thus students might not be guaranteed exposure to modes of computational thinking most relevant to their own disciplines and interests.

d. **Recommendation**

i. Given the challenges above, the working group declines to endorse this option.

2. **A subject that is designed for a major, or subjects that are designated as suitable for a major:**

a. **Description:**

i. A department would either develop a computation course specific to the interests of their students or use a subject offered by another department.

ii. This approach would be similar to the current CI-M communication requirement, wherein specific skills are taught in the context of a major.

b. **Advantages:**

i. Students would have the opportunity to see computation in a context that is particularly appealing to them, thereby increasing the utility of learned concepts in future educational experiences.

ii. Embedding the introduction of computation within a specific field has the potential to increase student engagement, since it would appear in contexts of interest to the student.

c. **Challenges:**

i. Some coordination of offerings would be required, to ensure that subject offerings by individual departments were meeting the goals of the requirement. This would include ensuring that core fundamental concepts are covered by all such offerings at an appropriate depth and level. A structure such as SOCR (Subcommittee on the Communication Requirement) might be required to provide an ongoing MIT-wide perspective as this is accomplished.

ii. Departments wishing to offer instruction in focused and/or advanced computational thinking would need to either teach basic programming skills (requiring additional teaching resources) or require a prerequisite basic programming class thereby extending the real reach of the requirement.

iii. Some departments may not feel capable of, or have the resources to, develop specific computation courses. This could be addressed by providing additional resources, or by encouraging collaboration with departments with experience in teaching computation.

d. **Recommendation**

i. The working group believes that this option is worth considering in more detail.

3. **An interdepartmental computational thinking requirement followed by a disciplinary computational thinking class.**
    a. **Description**
        i. The first part of the requirement would be satisfied by a small group of 6-unit, probably half-term subjects that give an introduction to computational thinking (offered at different levels--from no prior knowledge to advanced programming). Students would not be able to waive this requirement, so even students with extensive prior programming experience would take a class aimed at their level of background.
        ii. The second part of the requirement would be satisfied by an upper level class, which could be part of a department's requirements, that uses computational thinking as a part of the modes of thinking and learning in a specialized field.
        iii. The second requirement could be a stand-alone 6-unit class or as part of a larger unit subject that offers substantial computational thinking instruction.
        iv. The division between general and specific instruction would be comparable to the division between CI-H and CI-M classes in the communication requirement, while the ability to embed six-units of computational thinking within a larger class finds an analogy in certain lab courses that provide six units of lab credit within a 12 unit class.
    b. **Advantages**
        i. By offering an interdisciplinary subject followed by a discipline specified subject, students would acquire knowledge and modes of thinking that would be both flexible and narrowly tailored to a specific domain.
        ii. Skills in computational thinking in the interdisciplinary subject could be paired with classes with differing levels of programming experience creating a challenging but rewarding requirement for students of all backgrounds.
        iii. By requiring one or both parts of the requirement as a prerequisite, departmental subjects could be offered that can assume knowledge of computational algorithms and programming skills, allowing advanced topics to be offered more easily than they can be now.
    c. **Challenges**
        i. Although no department or unit would be required to create its own disciplinary subject, some departments may wish to do so but may be limited in their resources or their ability to develop such a class. Collaboration with other departments to develop appropriate subjects or using an existing subject offered by another department are possible solutions to this issue.

ii. The two-part requirement is sufficiently complex that additional advising and education about the requirement would be necessary.
iii. At least initially, an oversight committee similar to SOCR would need to be created to determine whether subjects fulfill the goals and expectations of computationally rigorous thinking.

d. **Recommendation**
   i. The working group believes that this approach offers myriad advantages and the challenges are of a manageable size. It recommends that implementation details be carefully explored, including impact on overall student load, resource requirements, and other implications of creating and offering such subjects.

Mitigation of Total Requirements

All of the options that the working group listed above, which were the only choices we felt gave a worthwhile introduction to computational thinking, add 12 units of requirements to the existing MIT GIRs. Given how tightly the curriculum already constrains student choice, this would be challenging.

An option studied in the past would be to add an additional GIR, but allow students to select 6 of 7 GIR's (while allowing departments to specify a subset of the GIR's as required for students choosing to major in that department). This option was previously explored by the Silbey committee, and may merit revisiting. However, the working group notes that this would run counter to the desire to have all students acquire a command of computational modes of thinking.

A second option, that was much more appealing to the working group, is to use one of the two current REST requirements as a computation requirement. Many existing Courses already designate a required REST subject that teaches some or all of the modes of computational thinking described above. However, we note that many department degree requirements already "capture" one or two REST subjects for different reasons. Thus, a careful study on the impact of allocating a REST subject to computation on departmental degree requirements would be needed by a future implementation group.

An associated issue that will need careful consideration, if the decision to implement a computational requirement for all students is made, is the impact of adding a degree requirement to ABET accreditation of engineering departments. Many engineering departments use REST subjects or current departmental subject to meet the constraints of professional accreditation of degree programs; thus any proposed changes to institute requirements will need to carefully consider the impact of different options on the ability of departments to preserve their accreditation status.

Summary

The working group unanimously believes that computational thinking is an essential part of the educational experience for every undergraduate student at the Institute. This is based on the view that computational thinking provides a new intellectual mode of thought of relevance to virtually every intellectual discipline and that computational thinking requires and develops important modes of communication. These factors are in addition to the pragmatic advantages that computational tools might give at MIT or in professional life. The working group recommends that the Institute proceed with a consideration of mechanisms by which a computation requirement could be instituted for all undergraduate students, while addressing the impact adding an additional degree requirement would have on student load and the need to connect computational thinking to domain-specific contexts across different intellectual disciplines.

Submitted by:

- Eric Grimson, EECS, Chair
- Deepto Chakrabarty, Physics
- Michael Scott Cuthbert, Music and Theater Arts
- Peko Hosoi, Mechanical Engineering
- Caitlin Mueller, Architecture
- James Orlin, Sloan
- Troy Van Voorhis, Chemistry

Appendix:

The full charge to the committee follows:

Study Group on Algorithmic and Computational Thinking for MIT Undergraduates

Charge

For many years, at least since the 2004-2006 Taskforce on the Undergraduate Educational Commons chaired by Prof. Robert Silbey, various MIT faculty members have asked whether, and if so how, MIT should ensure that all its undergraduates learn algorithmic reasoning and computational thinking. To answer this question, we are charging a small group of faculty to conduct an in-depth study of what the phrases "algorithmic reasoning" and "computational thinking" mean in the context of the education of MIT's undergraduates across all five schools.

In conversation, many colleagues who have thought about this issue are clear in saying that these phrases should mean more than an introduction to programming languages. As a place to begin this study, we believe that computational thinking should encompass an intellectual framework, not just a skill. Phrases that were used in the Silbey report include "computational modes of analysis", "algorithmic reasoning", "data abstraction", "designing computational solutions to theoretical and practical problems", and "providing a computational paradigm for reasoning and problem solving." We are asking you to do a careful, deliberative assessment of what these and other phrases ("abstraction and complexity", "modularity and interfaces", "complexity of algorithmic solutions", "algorithmic paradigms") mean across MIT.

Questions that we would ask you to examine include:

1) How do faculty, students and alumni in different fields of endeavor, across the full breadth represented by MIT's five schools, use computational thinking? Is it an important mode of thinking in (for example) economics, policy formation, management, architecture, biology and biological engineering, chemistry and chemical engineering, and other disciplines?

2) What, if any, is the common intellectual framework that people across MIT employ when they speak of computational thinking and algorithmic reasoning? In what ways is diversity among the meanings of such phrases in different disciplinary contexts important?

3) To what extent are algorithmic reasoning and computational thinking already being taught? What fraction of our graduates, across all five schools, learn them in the course of meeting the explicit requirements of their majors? What fraction take a course that covers computational thinking even if not an explicit requirement of their majors? To what extent and in what ways do we already implicitly expect that

a broad spectrum of MIT undergraduates across many majors understand algorithmic and computational thinking by the time they graduate from MIT? When in their career at MIT do we expect students to learn computational thinking?

4) Should we acknowledge algorithmic and computational thinking as an explicit expectation of all our graduates? If yes, what is the rationale/case for this?

5) If yes, what are the key elements of algorithmic and computational thinking and what are the associated learning objectives and measurable outcomes for knowledge, skills and attitudes? How are they common across the broad spectrum of MIT undergraduates, and how do they differ? Across MIT, how are they relevant to solving problems and mastering endeavors?

6) If yes, does it matter when during their careers at MIT our students are exposed to computational thinking and algorithmic reasoning? What benefits would accrue from a uniform approach to teaching them and what might the downsides be? What benefits would accrue from discipline-specific approaches and what might the downsides be?

7) What are our peer institutions doing? Are there possible models outside MIT that merit our consideration?

As you start to formulate your answers to the questions above, we would ask that you develop a list of possible options for accomplishing the goals for the computational education of MIT undergraduates that you articulate, if these goals are not already being met. Please describe each such option as concretely as you can, including pros and cons, including which goals among those you articulate each option addresses, and including actionable next steps. Examples of options that you might consider include:

i) Modules, with or without online components, that could be incorporated within MIT's existing GIR subjects.
ii) New subjects or modules with no prerequisites, ranging in duration from one month to one semester, whose development and teaching may involve collaboration among departments and other academic units.
iii) A model for teaching computational thinking along the lines of how CI-M subjects teach communication, where each major can make discipline-specific choices for how to achieve overarching MIT-wide goals that you have articulated, via more advanced subjects or modules designed for students in the specific major.

The first two are examples of options where next steps would include curriculum development. For such options, we hope that you will provide preliminary examples of partial syllabi, with explanations of your rationales for the elements in these syllabi, and a sense of the (groups of) colleagues who might be asked to develop the curricula. That is, these are examples of options that we would hope you develop to

the point that the next step could be Dean Freeman pulling together people and resources for implementation. The third is an example of an option where your study might prompt some departments to initiate next steps, perhaps with support from Dean Freeman. All are examples of options where the next steps would include consideration by a broader group of faculty, including relevant faculty committees.

We are convinced that a deep study as described above is a key step toward evaluating whether or not changes to MIT's undergraduate curriculum and pedagogy are merited. Depending on your findings, your study may provide the foundation for subsequent advances in how we educate our students. We are asking you to focus on questions as above and on options with near-term actionable next steps. We hope that the answers that your study provides, together with any subsequent curriculum development that it prompts, will serve as valuable input to any future discussion of our GIRs.

We would ask that you set as your goal that by June 30, 2016 you have completed the majority of your work and reported your progress and your emerging conclusions to us, so that by that date we have a full understanding of what remains for you to do, and a firm late-summer or early September deadline for your report.

Sincerely,

Prof. Dennis Freeman
Dean for Undergraduate Education

Prof. Krishna Rajagopal
Chair of the MIT Faculty

———————————

Membership:

Eric Grimson, Chair, EECS
Deepto Chakrabarty, Physics
Michael Cuthbert, Music and Theater Arts
Peko Hosoi, Mechanical Engineering
Caitlin Mueller, Architecture
Jim Orlin, Sloan
Troy van Voorhis, Chemistry