Programming the Central Dogma: An Integrated Unit on Computer Science and Molecular Biology Concepts

Anna Ritz

Reed College

Biology Department

3203 SE Woodstock Boulevard

Portland, Oregon 97202-8199

aritz@reed.edu

ABSTRACT

Much of modern biology requires quantitative and computational skills for the proper analysis of large-scale datasets, and there is a recognized need for computational training in undergraduate biology programs. This experience report describes a four-week unit designed to introduce fundamental computer science concepts and molecular biology concepts in an integrated fashion. The unit serves as the first four weeks of an introductory course taught within the Biology Department at an undergraduate institution, and is designed to introduce computational thinking to non-computational science majors. Survey results reveal that the course has attracted students from all years (first years through seniors), the majority of students have been women, and students have large self-perceived learning gains in computer science concepts. The unit shows promise for engaging non-computational students through applications in introductory molecular biology. Materials are available at <http://www.reed.edu/biology/courses/bio131/resources.html>.

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**; **Computer science education**; • **Applied computing** → *Bioinformatics*;

KEYWORDS

Computational biology; introductory programming; undergraduate education; computational thinking

ACM Reference format:

Anna Ritz. 2018. Programming the Central Dogma: An Integrated Unit on Computer Science and Molecular Biology Concepts. In *SIGCSE'18: The 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, February 21–24, 2018*. ACM, NY, NY, USA. 6 pages. <https://doi.org/http://dx.doi.org/10.1145/3159450.3159590>

1 INTRODUCTION

Biology has seen an explosion of enormous, complex datasets that require computational methods to process, store, and analyze them.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCSE'18, February 21–24, 2018, Baltimore, MD, USA

© 2018 Copyright held by the owner/author(s).

ACM ISBN ISBN 978-1-4503-5103-4/18/02.

<https://doi.org/http://dx.doi.org/10.1145/3159450.3159590>

The number of publications that involve *bioinformatics*, loosely defined as research at the intersection of biology and computer science, is expected to grow by as much as 93% by 2025 [15]. A current challenge in biology education is to introduce quantitative skills into the curriculum that engages students who may otherwise be unprepared for quantitative and computational reasoning [24]. There is a recognized need for computational biology courses as part of scientific training, and recent efforts have established communities of educators focused on bioinformatics within biology [8] and computer science [2]. However, many existing computational biology courses are typically intended for graduate or upper-level undergraduate students [17].

This experience report aims to introduce computational concepts to biology majors early in their undergraduate education. The overall goal of the four-week unit is to provide a crash course on introductory programming that is intertwined with biology concepts. The computational problems posed in the three main assignments are critical in computational biology, and capture much of the programming concepts necessary for first-year programmers. While this is not a rigorous computer science course, the immediate application to real data aims to engage non-computational students.

1.1 Related Work

Computational biology education has been a focus for over twenty years [1, 17], and there is a growing amount of training materials for biologists [10, 21, 23]. Workshops developed by Software Carpentry [28] and Data Carpentry [25] have helped train current practitioners as well as students entering the field. Pavel Pevzner and Phillip Compeau have developed a six-course Coursera series on Bioinformatics Algorithms, which uses a textbook combined with Rosalind, a web-based platform for learning about computational biology by problem solving [6].

In addition to online resources, computational biology training has appeared at the undergraduate level across many institutions. Computational biology is a common elective course in computer science programs, where it is typically presented as an upper-level algorithms course on biological sequence analysis. These courses draw on textbooks that describe algorithms on strings and their applications [5, 12, 14]. The Coursera series on bioinformatics algorithms grew out of Pavel Pevzner's introductory courses at University of California, San Diego [5, 6, 14], which has been a leading example of undergraduate training in computational biology.

Within undergraduate biology programs, computational biology courses have a notably different flavor. Computational topics in biology courses aim to teach students how to use well-established

tools such as BLAST for local sequence alignment and Clustal for multiple sequence alignment [17]. Recent textbooks written by biologists [13] or co-authored by biologists and computer scientists [4] are promising resources for including more programming concepts into undergraduate biology courses.

Many computational biology topics covered in tool-based and algorithmic-focused classes are similar – they include sequence alignment, motif finding, genome assembly, and phylogenetic reconstruction – but the courses are usually intended for intermediate to advanced undergraduate students [17]. There are notable exceptions, however, and a well-known example is Harvey Mudd's introductory BioCS1 course, which has successfully placed equal emphasis on biology and computer science topics at an introductory level [9, 16]. The course is team taught with computer science and biology faculty, which is the ideal scenario if there is enough faculty support at the institution, and materials have been adapted by other institutions. This report offers suggestions for a smaller unit that introduces computational thinking within an introductory biology curriculum.

1.2 Audience and Academic Environment

This four-week unit is the first component of an *Introduction to Computational Biology* course taught in the Biology Department at Reed College, a primarily undergraduate institution in Portland, Oregon. This course is separate from the recently-established Computer Science program at Reed, yet it is grounded in computer science concepts and taught by a computer scientist affiliated with the Biology Department. *Introduction to Computational Biology* was developed based on initial lectures from a Computational Thinking course from Brown University's Department of Computer Science [20], and uses the Compeau and Pevzner textbook [5] along with the companion Rosalind website [22].

Introduction to Computational Biology requires no programming or computer science experience, but students are expected to have completed first-semester introductory biology or equivalent. In addition to three hours of lecture, there is a two hour computational laboratory with structured labs that lead into the relevant assignment. The professor and an undergraduate TA are available during the lab sections. To-date, 49 students have completed the course over three semesters. Students range from first years to seniors and include both biology and non-biology majors (Figure 7). There are typically a few students who are familiar with programming and computer science concepts (e.g. they have complete CS1 or equivalent). More details about the introductory course are presented in Section 3.

1.3 The Central Dogma

The Central Dogma is a concept coined by Francis Crick that, at its most fundamental level, describes the transfer of information within a cell [7]. The genome is comprised of a double-stranded molecule called DNA, carefully organized and packed into chromosomes. Our DNA is nearly identical in every one of our trillions of cells, yet cells perform dramatically different functions. The information from our genome is selectively transferred into the synthesis of proteins, molecules that carry out complex biochemical reactions that result in a cellular response (such as grow and divide, recycle

Figure 1: Simplified View of the Central Dogma.

components, or move). This transfer of information typically requires a single-stranded molecule called RNA. The phrase “DNA makes RNA makes protein,” while over-simplified, summarizes the transfer of information that has been collectively defined as the Central Dogma of molecular biology (Figure 1).

2 CONTENT

The transfer of information summarized by the Central Dogma can be formalized as straightforward computational problems that introduce fundamental programming concepts. The unit is centered around eight computational problems that are directly related to sequence analysis (Table 1), which are described in further detail in the following subsections.

2.1 Week 1: Introduction to Functions

The first week of the unit does not begin with Python programming – it begins with Excel. Nearly every student has worked with spreadsheets; the point of the first few lectures is to reveal that they already know what a function is. The first assignment cements the following concepts within a spreadsheet environment: (a) variables and assignments, by referencing values by their cell location (e.g., A1), (b) types, by formatting cells as numeric or strings, and (c) using built-in functions such as LEN and SUM, and learning how to look up functions in an API. In addition to basic calculations, the first assignment aims to convey the difficulty of working with strings in Excel. Questions that involve standard Excel functions include calculating the length of a string, concatenating strings, removing characters from a string, and counting the number of occurrences of a character in a string. The last question in the assignment poses the first computational problem:

Problem 1 (HAM). Count the number of character differences between two strings s_1 and s_2 of equal length.

This problem is commonly known as computing the Hamming Distance [5], and appears at multiple points throughout the course. Students complete functions in the blue cells in Figure 2, and are assessed based on whether the function is correct and updates when different characters are entered into the green cells. Extensions of this problem include shifting the strings by prepending spaces to one of s_1 or s_2 (Figure 2 bottom), providing the optimization formulation of Problem 1:

Problem 2 (HAM-SHIFT). Find a shifted alignment by prepending spaces that minimizes the number of character differences between two strings s_1 and s_2 .

Problem 2 describes a variant of a sequence alignment problem, namely global ungapped alignment. Sequence alignment is covered in the middle of the semester; when gaps are allowed within the

Problem	Programming Concepts	Rosalind
<i>In Excel</i>	Variables & Assignments	
1 HAM	Variable Types	
2 HAM-SHIFT	Built-in Functions String Manipulation	
<i>In Python</i>	Strings, Ints, and Floats	
3 COUNT	Iteration	<i>DNA</i>
1 HAM	Conditionals	<i>HAMM</i>
4 REVCMP	User-Defined Functions	<i>REVC</i>
5 TRANSCRIBE	Lists	<i>RNA</i>
6 SPLICE	Slicing	<i>SPLC</i>
7 TRANSLATE	Program Organization	<i>TRANSLATE</i>
8 DOGMA	Dictionaries (advanced)	

Table 1: List of Computational Problems. Last column denotes most similar problem from Rosalind's Bioinformatics Stronghold Track [22].

strings, this problem becomes a classic motivation for dynamic programming. The goal of the first assignment is to convey that, while Excel is powerful for numeric data, it is a pain for text manipulation.

2.2 Week 2: String manipulation

Lectures from the end of Week 1 begin teaching basic Python programming concepts, starting with using the interpreter as a calculator. The second assignment includes a Python skeleton script where students “fill in holes” by working through directions in the comments. This includes standard programming assignments that introduce expressions and assignments, types (ints, floats, and strings), lists and indexing into strings and lists, and iteration.

During lectures, students begin manipulating toy DNA sequences. They learn about the double-stranded DNA molecule, which is comprised of four nucleotides (for simplicity, we will keep them as A/C/G/T). They first implement a customized LENGTH code snippet that uses a counter to compute the length of the string. They then learn about conditionals, which allows students to count the frequency of nucleotides in a string of DNA:

11	Calculate Hamming distance between two strings									
12	The Hamming distance between two strings is the number of characters that are different.									
13	Write a function to calculate the Hamming distance between A and B.									
14										
15	string A:	G	C	T	T	A	C	A		
16	string B:	G	T	C	T	T	A	C		
17										(Optional blue cells that may be useful)
18										
19	Hamming(A,B) =									

Alignment X:	G	C	T	T	A	C	A			
			G	T	C	T	T	A	C	
Hamming(X) =										

Figure 2: Hamming Distance Problem in Excel.

Figure 3: DNA Reverse Complement Problem.

Problem 3 (COUNT). Given a string $s \in \{A, C, G, T\}^*$, count the number of A's, C's, G's and T's.

The final problem in this assignment is related to the fact that DNA is double-stranded and nucleotides have an affinity for A/T pairs and C/G pairs. Further, there is a notion of directness for a strand of DNA, typically referred to as the 5' and 3' ends (and always read in the 5'-to-3' direction). Thus, given a string s of DNA in the 5'-to-3' direction, students write code to compute the reverse complement of s , which will also be in the 5'-to-3' direction (Figure 3):

Problem 4 (REVCMP). Given a string $s \in \{A, C, G, T\}^*$ (assuming the 5'-to-3' direction), return the reverse complement of s (in the 5'-to-3' direction).

The solution to this problem can be solved in two pieces: (a) creating a string comprised of the complement of each letter (e.g., A matched with T, C matched with G), and (b) reversing the contents of the string. This provides an opportunity to introduce functions, and establish a modular solution to the problem. By this point, students will gain familiarity with simple Boolean operations for string equality. As demonstration of their one-week progress, students can re-solve Problem 1 for any two strings of equal length.

2.3 Week 3: Transcription

The second half of the unit is designed to reinforce concepts learned in the first two weeks through small programming problems, while introducing more details about the Central Dogma. As shown in Figure 1, there are two main steps involved in the transmission of information from DNA to proteins. The first step converts the contents stored in DNA into RNA, a single-stranded nucleic acid that serves as an intermediary between DNA and proteins. DNA and RNA are structurally similar molecules, with notable differences in the type of ribose used as the “backbone” of the nucleic acid (deoxyribose in DNA vs ribose in RNA), and the fact that RNA is comprised of A/C/G/U rather than A/C/G/T. Pieces of DNA are discretized into regions called *genes*, which may be hundreds to thousands of nucleotides long. Genes are transcribed into RNA molecules by proteins that form a complex machinery, which will not be described here. While students learn about this process (especially the directionality of the process), the computational problem for transcribing a segment of DNA to RNA is straightforward (Figure 4):

Problem 5 (TRANSCRIBE). Given a string $s \in \{A, C, G, T\}^*$, return the transcribed RNA.

Figure 4: Transcription and RNA Processing Problem.

While RNA is commonly thought as the intermediary for producing proteins, recent work has revealed a diverse set of functions for RNA. The RNA molecules that code for proteins must be further processed into messenger RNA (mRNA) before they can move on to the next step of the Central Dogma. Genes that are encoded in DNA have small segments that describe parts of the proteins, called *exons*, separated by much larger segments that contain irrelevant sequence to the protein, called *introns*. Machinery must splice out the intronic regions of the transcribed RNA (Figure 4). While students learn about RNA processing (including other components of the messenger RNA), they can solve the following problem:

Problem 6 (SPlice). Given a string $s \in \{A, C, G, U\}^*$ and a list of exon and intron starts, return the mRNA comprising only the exons.

This problem can be complicated by the structure of the inputs. The current form of the assignment provides two lists of equal length: one list denotes exon start indices and the other list denotes intron start indices. As a warm-up, students are given a hypothetical gene 'eeeeeeeeiiiiiiiiiiii' with coordinates that should splice out (remove) the *i* characters in the string. Students can also calculate basic statistics about each gene that includes the number of exons, the average exon length, and the average intron length.

2.4 Week 4: Translation

In the final week of the unit, the focus shifts to the underlying molecular biology as we discuss the second step of the Central Dogma, which converts mRNA into protein, which are comprised of amino acids. During translation, mRNA is partitioned into non-overlapping substrings of length three called *codons*, where each codon maps to one of 20 possible amino acids (Figure 5). Amino acids are attached to a growing protein by “reading” the mRNA sequence in groups of three letters.

The first computational question we discuss is *why are codons three nucleotides long?* This is a classic information theory question: with an alphabet of four nucleotides, a single-letter code would only describe four items, a two-letter code would describe $4^2 = 16$ items, and a three-letter code would describe $4^3 = 64$ items. There are 20 amino acids (plus a signal for “stop”), so one needs the three-letter code to distinguish them. As a result, multiple codons map to the same amino acid. This redundancy has a pattern of variance in the third nucleotide due to the physical structure of the translation mechanism.

The computational problem of translating an mRNA to protein is formulated as follows:

Problem 7 (TRANSLATE). Given a string $s \in \{A, C, G, U\}^*$ (where $\text{mod}(\text{length}(s), 3) = 0$) and a codon table, return the protein described by the codons.

Figure 5: Translation Problem. Circular codon table adapted from [18].

The assignment includes a `codon_to_aa()` function that maps each 3-letter codon to the appropriate amino acid. The last task of the assignment is to put the entire pipeline together:

Problem 8 (DOGMA). Given a string $s \in \{A, C, G, T\}^*$ in the 5'-to-3' direction with a list of exon and intron starts, return the protein.

2.5 Case Study: Cancer Genes

Public repositories contain the sequences that comprise the human genome, and have been extensively revised since the sequenced genome was announced in the early 2000s. The UCSC Genome Browser is one of the most popular repositories, offering a searchable database that annotates genetic segments and a programmatic API [26]. Students are able to compute the solutions for Problems 3–8 on real gene sequences, meaning that the students begin working with real data at about the same time that they learn their first python programming concepts. As a case study, the last problem is framed as one that transcribes and translate cancer genes.

Cancer is a disease characterized by mutations at the DNA (gene) level, and many genes that are related to cancer have been studied for decades [27]. SRC (pronounced “sark”) is one such gene whose overexpression (in terms of RNA abundance) is associated with colon, breast, ovarian, and lung cancers, among many others [11]. This gene is over 22,000 nucleotides long, containing 13 exons that create a 1,611 nucleotide long mRNA sequence. This sequence is transcribed into a protein with 546 amino acids and one “stop” codon. The UCSC Genome Browser contains information about the DNA sequence, the mRNA sequence, and the protein sequence, as well as the exon and intron starts. Students are given all these pieces of information, and require their code to contain tests that evaluate whether the computed mRNA matches the real mRNA sequence and whether the computed protein matches the real protein sequence. Students are also provided with hypothetical genes for testing on a smaller scale, and require outputs to be printed to screen (Figure 6).

2.6 Extensions for Intermediate Programmers

This unit can be understandably slow-paced for students who are familiar with some programming. However, there are points that can be extended for more challenging problems. First, students are encouraged to write user-defined functions for standard built-in functions such as `len()` and `reverse()` for practice; students

```

Dataset: testPos
DNA: CCCATGGTCGGGGGGGGGAGTCCATAACCC
Num exons: 2
strand: +
RNA (from file): AUGGUCAGUCCAUA
peptide (from file): MVSP*

Computed mRNA: AUGGUCAGUCCAUA
mRNA matches!

Computed Peptide: MVSP*
Peptide matches!

```

Figure 6: Output from Problem 8 for a test gene.

Figure 7: Number of students by year, gender, and type of major.

end up adopting more built-in functions in the later part of the course. Second, students can write their own `codon_to_aa()` function using dictionaries. Finally, an advanced assignment is to extend Problem 8 for handling genes on the *negative strand* of a DNA molecule. The UCSC Genome Browser has standardized the format of storing gene sequences, returning the negative strand (in the 5'-to-3' direction) but always reporting exon boundaries according to the *forward* strand of the DNA. Students can handle the negative strand by re-mapping the exon boundaries (and reversing the order of the exons) before solving Problem 8, but it requires careful thought. This subtlety introduces students to both the complexity of standardization and file processing skills.

3 CONTEXT WITHIN A COURSE

The remaining components of this course use elements of the widely-used bioinformatics algorithms textbook by Compeau and Pevzner [5]. The units in the class are framed by the biological motivations within each chapter, but are adapted for an introductory audience (rather than an intermediate or upper-level undergraduate audience). They include:

- **Identifying the origin of DNA replication** by searching for frequently-occurring strings of length k (k -mers).

- **Finding DNA-binding protein motifs** with a greedy algorithm.
- **DNA assembly** by constructing a de Bruijn graph and identifying Eulerian cycles.
- **Sequence alignment** by implementing dynamic programming.

Each of the topics above has an associated programming assignment, where students implement basic aspects of the content. The last two weeks of the course are structured around final projects. Projects are wide-ranging, but many have involved extensions of the Central Dogma unit. For example, single-stranded RNA molecules tend to fold in on itself due to the A/U and C/G pairing; and intermediate programmers have successfully implemented the Nussinov dynamic program to compute the optimal number of nucleotide pair matchings given a string [19].

4 STUDENT RESPONSES

Student response from the *Introduction to Computational Biology* course evaluations has been positive overall. However, as one of the few introductory science courses with a small number of students (typically 15-20), these responses may be inflated due to the structure of the course and the individualized attention, rather than the effectiveness of the course itself. Further, the student evaluations were about the entire course, rather than this four-week unit. Thus, all 49 students who have completed the course to-date were anonymously surveyed to gather responses related to learning gains and preparation for the remainder of the course.¹ While there are established issues with surveys that assess undergraduate self-reported gains [3], the goal here was to identify changes in *self-perceived* learning gains, which are associated with a confidence in the subject.

Twenty-one students completed the survey, reflecting the diversity by year and gender (Figure 7). Before the start of the course, the majority of students were comfortable with molecular biology concepts, and the majority of students were unfamiliar with computational skills (Figure 8). Self-perceived learning gains in biology were not large, which is expected since students believed they had background in the subject before taking the course. This is also reasonable given the large proportion of juniors and seniors in the course (Figure 7). Strikingly, self-perceived gains in computational concepts are very large, especially when aggregated across all concepts (Figure 9).

Students were also asked about the usefulness of the assignments in preparation for the rest of the course. The assignments were listed as HW1 (Excel), HW2 (Python Practice) and HW3 (Central Dogma), which was further subdivided into transcription and translation. While the Excel homework was not as useful to students as the subsequent homework assignments, only one individual found HW1 to be not helpful and confusing (Figure 9). Written comments revealed that some students were unfamiliar with Excel, making the first assignment more difficult than intended. The usefulness of the transcription and the translation components of HW3 were listed as equally useful, and are not shown here.

¹This survey, disseminated as a Google Form, was deemed to be exempt by the Institutional Review Board.

Figure 8: Student responses about biology and CS concepts on a 5-point scale.

Figure 9: Self-perceived experience, learning gains, and assignment usefulness. First two plots summarize the data in Figure 8.

5 CONCLUSION

New biological technologies are producing enormous datasets, requiring significant training in computational and quantitative skills. While computational biology courses exist at institutions, they are usually offered within computer science as upper-level electives. Thus, there is a need to introduce biology majors to computational reasoning early in their undergraduate education. This introductory-level unit integrates computational problems with molecular biology concepts, providing biology students an immediate connection and meaning for programming. The unit may also be useful in cross-disciplinary courses, where it is important to get everyone on the same page as quickly as possible. Student responses indicate that the unit helped cement computer science concepts, and the assignments were useful preparation for the remainder of the course. Future work includes developing a tool to automatically pull down and process *any* UCSC gene for the unit, providing full flexibility for students to pursue genes related to diseases of interest. This will help produce more robust faculty development materials for others to easily adapt the four-week unit in their courses.

REFERENCES

- [1] Russ B. Altman. 1998. A curriculum for bioinformatics: the time is ripe. *Bioinformatics* 14, 7 (1998), 549–550.
- [2] Teresa K Atwood, Erik Bongcam-Rudloff, Michelle E Brazas, Manuel Corpas, Pascale Gaudet, Fran Lewitter, Nicola Mulder, Patricia M Palagi, Maria Victoria Schneider, Celia WG van Gelder, et al. 2015. GOBLET: the global organisation for bioinformatics learning, education and training. *PLoS computational biology* 11, 4 (2015), e1004143.
- [3] Nicholas A Bowman and Patrick L Hill. 2011. Measuring how college affects students: Social desirability and other potential biases in college student self-reported gains. *New Directions for Institutional Research* 2011, 150 (2011), 73–85.
- [4] Caroline St Clair and Jonathan E Visick. 2013. *Exploring Bioinformatics*. Jones & Bartlett Publishers.
- [5] Phillip Compeau and Pavel Pevzner. 2015. *Bioinformatics algorithms: an active learning approach*. Active Learning Publishers.
- [6] Phillip Compeau and Pavel A Pevzner. 2015. Life After MOOCs. *Commun. ACM* 58, 10 (2015), 41–44.
- [7] Francis Crick. 1970. Central dogma of molecular biology. *Nature* 227, 5258 (1970), 561–563.
- [8] Elizabeth Dinsdale, Sarah CR Elgin, Neal Grandgenett, William Morgan, Anne Rosenwald, William Tappich, Eric W Triplett, and Mark A Pauley. 2015. NIBLSE: A network for integrating bioinformatics into life sciences education. *CBE-Life Sciences Education* 14, 4 (2015), 1e3.
- [9] Zachary Dodds, Ran Libeskind-Hadas, and Eliot Bush. 2010. When CS 1 is biology 1: crossdisciplinary collaboration as CS context. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*. ACM, 219–223.
- [10] Berk Ekmekci, Charles E McAnany, and Cameron Mura. 2016. An introduction to programming for bioscientists: A python-based primer. *PLoS computational biology* 12, 6 (2016), e1004867.
- [11] Margaret C Frame. 2002. Src in cancer: deregulation and consequences for cell behaviour. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer* 1602, 2 (2002), 114–130.
- [12] Dan Gusfield. 1997. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press.
- [13] Steven Harold David Haddock and Casey W Dunn. 2011. *Practical computing for biologists*. Sinauer Associates Sunderland, MA.
- [14] Neil C Jones and Pavel Pevzner. 2004. *An introduction to bioinformatics algorithms*. MIT press.
- [15] Jonghoon Kang, Seyeon Park, Aarya Venkat, and Adarsh Gopinath. 2015. Quantitative analysis of the trends exhibited by the three interdisciplinary biological sciences: biophysics, bioinformatics, and systems biology. *Journal of microbiology & biology education* 16, 2 (2015), 198.
- [16] Ran Libeskind-Hadas and Eliot Bush. 2013. A first course in computing with applications to biology. *Briefings in bioinformatics* 14, 5 (2013), 610–617.
- [17] Alejandra J Magana, Manaz Taleyarkhan, Daniela Rivera Alvarado, Michael Kane, John Springer, and Kari Clase. 2014. A survey of scholarly literature describing the field of bioinformatics education and bioinformatics educational research. *CBE-Life Sciences Education* 13, 4 (2014), 607–623.
- [18] Kenneth R Miller and Joseph S Levine. 2006. *Prentice hall biology*. Boston, MA: Pearson Prentice Hall. Retrieved on February 4 (2006), 2014.
- [19] Ruth Nussinov and Ann B Jacobson. 1980. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences* 77, 11 (1980), 6309–6313.
- [20] Laura Pappano. 2017. Learning to Think Like a Computer. *The New York Times: Education Life* (2017).
- [21] Jeffrey M Perkel. 2015. Pick Up Python. *Nature* 518, 7537 (2015), 125.
- [22] Pavel Pevzner. 2012. Rosalind. (2012).
- [23] Ashley Shade and Tracy K Teal. 2015. Computing workflows for biologists: a roadmap. *PLoS biology* 13, 11 (2015), e1002303.
- [24] Susan Singer and Karl A Smith. 2013. Discipline-Based Education Research: Understanding and Improving Learning in Undergraduate Science and Engineering. *Journal of Engineering Education* 102, 4 (2013), 468–471.
- [25] Tracy K Teal, Karen A Cranston, Hilmar Lapp, Ethan White, Greg Wilson, Karthik Ram, and Aleksandra Pawlik. 2015. Data carpentry: workshops to increase data literacy for researchers. *International Journal of Digital Curation* 10, 1 (2015), 135–143.
- [26] Cath Tyner, Galt P Barber, Jonathan Casper, Hiram Clawson, Mark Diekhans, Christopher Eisenhart, Clayton M Fischer, David Gibson, Jairo Navarro Gonzalez, Luvina Guruvadoo, et al. 2016. The UCSC Genome Browser database: 2017 update. *Nucleic acids research* 45, D1 (2016), D626–D634.
- [27] Bert Vogelstein and Kenneth W Kinzler. 2004. Cancer genes and the pathways they control. *Nature medicine* 10, 8 (2004), 789.
- [28] Greg Wilson. 2006. Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive. *Computing in Science & Engineering* (2006).