# SlimArc-PRAXIS_17-22 cid:image003.jpg@01D1B73F.29BF98A0

# 

# Computer Science (5652)

|  |  |  |  |
| --- | --- | --- | --- |
| *Test at a Glance* | | | |
| Test Name | Computer Science | | |
| Test Code | 5652 | | |
| Time | 180 minutes | | |
| Number of Questions | 100 | | |
| Format | The test consists of a variety of short-answer questions such as selected-response questions (where you select one or multiple answer choices, depending on what the question asks for), questions where you enter your answer in a text box, and other types of questions.  You can review the possible question types in chapter 2, Familiarize Yourself with Test Questions. | | |
| Test Delivery | Computer delivered | | |
|  | Content Categories | Approximate Number of Questions | Approximate Percentage of Examination |
| Impacts of Computing | 15 | 15% |
| Algorithms and Computational Thinking | 25 | 25% |
| Programming | 30 | 30% |
| Data | 15 | 15% |
| Computing Systems and Networks | 15 | 15% |

**About This Test**

The *Praxis* Computer Science test is designed to assess the computer science knowledge and competencies necessary for a beginning teacher of secondary school computer science. Examinees have typically completed   
a bachelor’s program with an emphasis in computer science or computer science education.

The examinee will be required to understand and work with computer science concepts, use algorithms and computational thinking, work with code, manipulate data, and demonstrate knowledge of computing systems and networks.

The test is not designed to be aligned with any particular computer science curriculum, but it is intended to be consistent with the recommendations of national studies on computer science education, such as the Code.org   
*K-12 Computer Science Framework* (2016), the Computer Science Teachers Association (CSTA) *K-12 Computer Science Standards* (2017), and the International Society for Technology in Education (ISTE) *Standards for Computer Science Educators* (2011).

This test may contain some questions that do not count toward your score.

# Topics Covered

Representative descriptions of topics covered in each category are provided below.

1. Impacts of Computing  
   1. Understands and applies knowledge of impact   
       of, obstacles to, and effects of computing
2. Understand computing as a way of expressing creativity, solving problems, enabling communication, and fostering innovation in a variety of fields and careers
3. recognize that computers can be used to showcase creativity
4. recognize the benefits of using computers to solve problems
5. provide examples of how computers enable communication and collaboration
6. provide examples of how computers foster innovation
7. Know the obstacles to equal access to computing among different groups and   
   the impact of those obstacles
8. identify obstacles to equal access   
   to computing among different groups (e.g., groups defined by gender, socioeconomic status, disability/accessibility needs) and   
   the impact of those obstacles
9. identify factors that contribute to the digital divide
10. match obstacles to equal access with effective solutions
11. Understand beneficial and harmful effects of computing innovations and the trade-offs between them
12. analyze computing innovations in terms of their social, economic, and cultural impacts, both beneficial and harmful
13. identify trade-offs between beneficial and harmful effects of computer innovations
    1. Understands and applies knowledge of issues   
        regarding intellectual property, ethics,   
        privacy, and security in computing
14. Know different methods of protecting intellectual property rights and the   
    trade-offs between them in a variety of contexts (e.g., Creative Commons, open source, copyright)
15. using correct vocabulary, describe   
    how different methods of protecting intellectual property rights work
16. given a context, identify appropriate methods of protecting intellectual property rights
17. identify and compare trade-offs between different methods of protecting intellectual property rights
18. Understand ethical and unethical computing practices and their social, economic, and cultural implications
19. identify ethical and unethical computing practices in context
20. describe the social, economic, and cultural implications of ethical and unethical computing practices
21. identify the conditions under which a given computing practice is ethical or legal
22. Know privacy and security issues regarding the acquisition, use, and disclosure of information in a digital world
23. using correct vocabulary, describe privacy and security issues
24. in context, identify appropriate strategies to safeguard privacy and ensure security
25. describe trade-offs between local and cloud-based data storage
26. identify methods that digital services use to collect information about users
27. Algorithms and Computational Thinking
    1. Understands and applies knowledge of   
        abstraction, pattern recognition, problem   
        decomposition, number base conversion,   
        and algorithm formats
28. Understand abstraction as a foundation   
    of computer science
29. identify, create, or complete the   
    correct ordering, from low to high,   
    of an abstraction hierarchy
30. identify abstractions in context
31. identify details that can be removed from a solution in order to generalize it
32. Know how to use pattern recognition, problem decomposition, and abstraction   
    to develop an algorithm
33. given a table of values or other data source, identify the patterns in the data and identify algorithms that could produce the patterns
34. identify components that could be part of an algorithm to solve a problem
35. identify actions and actors when decomposing a problem
36. identify appropriate decomposition strategies
37. Understand number base conversion and binary, decimal, and hexadecimal number systems
38. convert between number bases
39. analyze and compare representations   
    of numbers in different bases
40. Understand how to develop and analyze algorithms expressed in multiple formats (e.g., natural language, flowcharts, pseudocode)
41. interpret diagrams that describe algorithms, given an explanation of the symbols used
42. compare algorithms written in multiple formats
43. trace and analyze algorithms written in different formats
44. identify correct sequencing of steps in an algorithm and errors in sequencing
    1. Understands and applies knowledge of   
        algorithm analysis, searching and sorting   
        algorithms, recursive algorithms, and   
        randomization
45. Be familiar with the limitations of computing in terms of time, space, and solvability as well as with the use of heuristic solutions that can address these limitations
46. identify and compare algorithms that   
    are linear, quadratic, exponential, or logarithmic
47. recognize the existence of problems that cannot be solved by a computer
48. in context, identify factors that prevent   
    a problem from being solvable
49. identify situations where heuristic solutions are useful
50. in context, identify space and time limitations of computational solutions   
    to problems
51. Understand searching and sorting algorithms; can analyze sorting algorithms for correctness and can analyze searching algorithms for correctness and efficiency
52. trace algorithms and predict output and intermediate results
53. calculate the number of comparisons required for linear and binary search algorithms
54. Understand simple recursive algorithms (e.g., *n* factorial, sum of first *n* integers)
55. trace simple recursive algorithms
56. provide missing steps in incomplete simple recursive algorithms
57. identify parts of a recursive algorithm (e.g., base or stopping condition, recursive call)
58. identify errors in simple recursive algorithms
59. identify an iterative algorithm that is equivalent to a recursive algorithm
60. Be familiar with the use of randomization   
    in computing
61. identify appropriate uses of randomization in a variety of applications
62. identify the difference between random and pseudorandom numbers
63. Programming
    1. Understands and applies knowledge of   
        programming control structures, standard   
        operators, variables, correctness, extensibility,   
        modifiability, and reusability
64. Understand how to write and modify computer programs in a text-based programming language
65. describe what a program does or be able to choose the code segment that correctly implements a given intended purpose
66. identify missing code in a code segment with a stated intended purpose
67. place statements in appropriate order   
    to create a correct program
68. identify how changing one part of a code segment will affect the output
69. Understand how to analyze computer programs in terms of correctness
70. trace code and indicate the output printed or the value of variables after code segment execution
71. indicate the inputs that produce given outputs for a code segment
72. describe what a program does or choose the code segment that correctly implements a given intended purpose
73. identify valid preconditions and postconditions
74. compare two code segments or algorithms
75. identify the type of error produced by   
    a code segment (i.e., syntax, runtime, compile-time, overflow, round-off, logic)
76. identify errors in incorrect code and changes that can be made to correct them
77. Know the concepts of extensibility, modifiability, and reusability
78. identify the meaning of the terms
79. identify functionally equivalent statements or code segments that   
    differ in one of these three ways
80. identify situations where the use of constants or variables would be preferred over hard-coded values
81. identify opportunities for parameterization
82. choose code that improves on given code by making it more extensible, modifiable, or reusable
83. identify changes that would improve   
    a given code segment
84. Understand the three basic constructs used in programming: sequence, selection, and iteration
85. trace code and indicate the output printed or the value of variables after code segment execution
86. indicate inputs that produce given outputs for a code segment
87. describe what a program does or choose the code segment that correctly implements a given intended purpose
88. identify missing code in a code segment with a stated intended purpose
89. identify equivalent statements or code segments
90. identify the three constructs when used in code
91. identify which of the constructs are needed to implement given functionality
92. convert code that does not use iteration to equivalent code that uses iteration
93. Understand how to use standard operators (i.e., assignment, arithmetic, relational, logical) and operator precedence to write programs
94. trace code and indicate the output displayed or the value of variables after code segment execution
95. indicate inputs that produce given outputs for a code segment
96. describe what a program does or choose the code segment that correctly implements a stated intended purpose
97. identify missing code in a code segment with a stated intended purpose
98. identify equivalent statements or code segments
99. place statements in appropriate order to create a correct program
100. use Boolean algebra to identify equivalent Boolean expressions
101. write a Boolean expression equivalent to given code, or identify code equivalent to a given Boolean expression or English description
102. identify the correct implementation of a given formula, including formulas with fractions
103. evaluate expressions that include arithmetic operations
104. Understand how to use variables and a variety of data types
105. identify variables and data types (e.g., integers, floating point, string, Booleans, arrays/lists)
106. identify the need for type conversion
107. trace code and indicate the output printed or the value of variables after code segment execution
108. indicate the inputs that produce given outputs for a code segment
109. describe what a program does or choose the code segment that correctly implements a stated intended purpose
110. identify missing code in a code segment with a stated intended purpose
111. identify equivalent statements or code segments
112. place statements in appropriate order   
     to create a correct program
113. describe the difference between integer and floating point numeric data types
114. describe the difference between integer and floating point division
115. describe the benefits of the use of each data type
116. distinguish between global and local scope
117. identify the most appropriate data type in a given context
118. identify the correct sequence of string operations to produce a given output
     1. Understands and applies knowledge of   
         procedures, event-driven programs, usability,   
         data structures, debugging, documenting and   
         reviewing code, libraries and APIs, IDEs, and   
         programming language paradigms, including   
         object-oriented concepts
119. Understand how to write and call procedures with parameters and return values
120. trace code and indicate the output printed or the value of variables after code segment execution
121. indicate inputs that produce given outputs for a code segment
122. describe what a program does or choose the code segment that correctly implements a stated intended purpose
123. identify missing code in a code segment with a stated intended purpose
124. identify equivalent statements or code segments
125. place statements in appropriate order   
     to create a correct program
126. trace code when references to objects and arrays are passed to procedures
127. trace code that includes nested procedure calls
128. Know the concepts of event-driven programs that respond to external events (e.g., sensors, messages, clicks)
129. trace code and indicate the output printed or the value of variables after code segment execution
130. indicate inputs that produce given outputs for a code segment
131. describe what a program does or choose the code segment that correctly implements a stated intended purpose
132. identify missing code in a code segment with a stated intended purpose
133. identify possible errors due to asynchronous events
134. identify aspects of concurrency in event-driven programming
135. Be familiar with usability and user experience (e.g., ease of use and accessibility)
136. identify code that improves on given code in terms of usability or user experience
137. identify meaningful error messages
138. identify features that improve accessibility
139. Be familiar with dictionaries/maps, stacks, and queues
140. identify a data structure based on a description of behavior or appropriate use
141. given goals, constraints, or context, identify the most appropriate data structure
142. trace code that uses a particular data structure
143. Understand how to use debugging techniques and appropriate test cases
144. identify which test cases are most useful for given code
145. differentiate between different types of errors (e.g., overflow, round-off, syntax, runtime, compile-time, logic)
146. describe useful debugging techniques (e.g., where to put print statements)
147. differentiate between empirical testing and proof
148. identify errors in code and solutions   
     to those errors
149. Be familiar with characteristics of well-documented computer programs that are usable, readable, and modular
150. identify characteristics of good documentation
151. identify good and poor documentation practices in context
152. Be familiar with techniques to obtain and use feedback to produce high-quality code (e.g., code reviews, peer feedback, end user feedback)
153. identify situations in which each of the three listed techniques are useful
154. Know how to use libraries and APIs
155. identify correct call(s) and use of return values given an API definition
156. identify reasons to use or not use libraries in place of writing original code
157. identify applications (e.g., math libraries, random number generation) that use APIs
158. Understand programming techniques to validate correct input and detect incorrect input
159. identify effective input data validation strategies
160. compare data validation (proper range and format) and data verification (e.g., password verification)
161. identify improvements to code for which data validation is required
162. Be familiar with the features and capabilities of integrated development environments (IDEs)
163. identify components of IDEs
164. identify benefits and drawbacks of using IDEs
165. identify the costs and benefits of context editors
166. Be familiar with the differences between low- and high-level programming languages
167. identify characteristics of low- and high-level languages
168. Be familiar with different programming paradigms
169. identify the terminology of procedural programming
170. identify the terminology of object-oriented programming
171. compare programming paradigms
172. Know object-oriented programming concepts
173. identify classes, instance variables,   
     and methods given a diagram
174. identify the benefits of inheritance   
     and encapsulation
175. identify distinctions between overloading and overriding
176. Be familiar with program compilation and program interpretation
177. identify differences between compilation and interpretation
178. identify differences between source code and object code
179. Data  
     1. Understands and applies knowledge of   
         digitalization, data encryption and decryption,   
         and computational tools
180. Understand bits as the universal medium for expressing digital information
181. perform calculations, using bits and bytes
182. determine the number of bits and bytes required to store a given amount of data
183. given the description of an encoding scheme, encode or decode data
184. describe lossy and lossless data compression
185. explain why binary numbers are fundamental to the operation of computer systems
186. Be familiar with concepts of data encryption and decryption
187. distinguish between encoding and encryption
188. identify trade-offs in the use of data encryption
189. Know how to use computational tools, including spreadsheets, to analyze data   
     in order to discover, explain, and visualize patterns, connections, and trends
190. transform data to make it more useful
191. identify specific data or characteristics of specific data that need to be removed or modified before an entire data set can be used
192. describe the use of spreadsheet operations (e.g., formulas, filters, sorts, charts, graphs) to analyze and visualize data
     1. Understands and applies knowledge of   
         simulation, modeling, and manipulation   
         of data
193. Be familiar with the use of computing   
     in simulation and modeling
194. describe questions that can be answered with a given simulation,   
     or explain what data and process are required in a simulation in order to answer a given question
195. trace code in a simulation context
196. identify missing code in a simulation context
197. identify the impact of changes to simulations (e.g., more or fewer variables, more or less data)
198. identify applications of simulation   
     and modeling
199. Be familiar with methods to store, manage, and manipulate data
200. use terminology and concepts of files and databases
201. identify measures of file size   
     (e.g., byte, kilo, mega, giga, tera, peta)
202. identify issues connected with the storage requirements of computing applications, including scale, redundancy, and backup
203. Be familiar with a variety of computational methods for data collection, aggregation, and generation
204. identify the benefits of working with publicly available data sets
205. identify the types of data generated by surveys and sensors
206. identify examples of crowdsourcing and citizen science
207. identify appropriate data-collection methods for a given context and purpose
208. Computing Systems and Networks  
     1. Understands and applies knowledge   
         of operating systems, computing systems,   
         communication between devices, and cloud   
         computing
209. Know that operating systems are programs that control and coordinate interactions between hardware and software components
210. identify hardware components and their functions
211. identify software components and their functions
212. identify common operating systems tasks
213. identify resource issues that have an impact on functionality
214. Be familiar with computing systems embedded in everyday objects (e.g., Internet of Things [IoT], ATMs, medical devices)
215. describe what an embedded system is
216. define what the IoT is and how it is used
217. describe how sensors are used in embedded systems
218. Know the capabilities, features, and uses   
     of different types of computing systems (e.g., desktop, mobile, cluster)
219. identify capabilities, features, and uses for each type of computer system
220. identify criteria to evaluate and compare computing systems
221. Be familiar with computers as layers of abstraction from hardware (e.g., logic gates, chips) to software (e.g., system software, applications)
222. identify appropriate abstraction layers for hardware and software components
223. Be familiar with the steps required to execute a computer program (fetch-decode-execute cycles)
224. describe what happens during fetch, decode, and execute, including the order of the steps in the cycle
225. Be familiar with trade-offs between local, network, and cloud computing and storage
226. identify advantages and disadvantages in terms of performance, cost, security, reliability, and collaboration
227. identify means of storing binary data
228. Be familiar with communication between devices
229. identify and compare wireless communication systems
230. identify and compare wired communication systems
231. identify and compare network types
     1. Understands and applies knowledge of   
         networks, including security issues and   
         the Web
232. Know components of networks
233. identify network hardware devices   
     and their functions
234. describe possible abstraction models   
     of networks
235. Be familiar with factors that have an impact on network functionality
236. define basic terminology   
     (e.g., bandwidth, load, latency)
237. estimate necessary bandwidth and   
     data size for a given situation
238. identify critical resources for a given situation
239. Be familiar with how Internet and Web protocols work
240. describe the purpose of protocols and identify common Internet and Web protocols
241. compare IPv4 and IPv6
242. identify and describe the basic parts   
     of a URL (e.g., protocol, subdomain, domain name, port, path)
243. describe the hierarchical structure of names in the domain name system (DNS)
244. describe the purpose and function of   
     IP addressing
245. identify how Internet protocols address reliability, redundancy, and error handling
246. Be familiar with digital and physical strategies for maintaining security
247. identify characteristics of strong passwords (e.g., length, bits per character)
248. identify digital and physical security strategies
249. identify trade-offs in the use of security measures (e.g., encryption, decryption, digital signatures and certificates)
250. Be familiar with concepts of cybersecurity
251. identify and define the five pillars of cybersecurity: confidentiality, integrity, availability, nonrepudiation, and authentication
252. Be familiar with the components that make up the Web (e.g., HTTP, HTML, browsers, servers, clients)
253. identify the uses of markup languages
254. identify the purposes of browsers, servers, and clients

# Code Segments

# Some stimulus material contains code segments written in pseudocode. The notation used in the pseudocode is described below.

PSEUDOCODE NOTATION

|  |  |
| --- | --- |
| **Explanation** | **Notation** |
| Assignment operator | ← |
| Arithmetic operators | + - / \* ^ %  Note that / indicates floating point division unless stated otherwise. |
| Relational operators | == < > ≤ ≥ ≠ |
| Logical operators | **and**  **or**  **not** |
| String concatenation operator | + |
| Boolean values | **true**  **false** |
| Null | **null** |
| Comments | // this is a single-line comment |
| Placeholder for missing code | For example,  /\* missing code \*/  /\* missing condition \*/ |
| Print  A comment is used where necessary to indicate if a line feed or blank is appended to the argument. | **print** arg |
| Data types | **boolean**  **char**  **double**  **float**  **int**  **int[]**  **int[][]**  **short**  **String** |
| Array initialization and reference | **int[]** a  {1, 2, 3}  **int** b[0..2]  {1, 2, 3}  **int[][]** c  a[0] |

|  |  |
| --- | --- |
| Conditional statements: Indentation and  **end if** statements are significant.  Example:  **if** ( x > 10 )  **print** "big number"  **else**  **print** "small number"  **end if** | **if** ( condition )  block of statements  **end if**  **if** ( condition )  block of statements  **else**  another block of statements  **end if** |
| Iterative statements: Indentation and **end** statements are significant. | **for** ( initialization; condition; increment )  block of statements  **end for**  **while** ( condition )  block of statements  **end while**  **do**  block of statements  **while** ( condition )  **repeat**  block of statements  **until** ( condition ) |
| Procedures: Indentation and **end** statements are significant.  The return type is indicated in the procedure header and is based on the value returned by the procedure or is **void** if the procedure does not return a value. | **int** procedureName( arg1,  arg2,  ... )  block of statements  **return** value  **end** procedureName  **void** procedureName ( arg1,  arg2,  ... )  block of statements  **end** procedureName |
| Classes | **class** className  variable declarations  procedures  **end** **class** className |
| Object-oriented keywords | **extends**  **new**  **public**  **private** |