#  cid:image003.jpg@01D1B73F.29BF98A0Computer Science (5652)

# Test at a Glance – ADVANCE COPY

|  |  |
| --- | --- |
| 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 | ApproximateNumber ofQuestions | ApproximatePercentage ofExamination |
| 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

Copyright © 2017 by Educational Testing Service. All rights reserved. ETS, the ETS logo, and PRAXIS are registered trademarks of Educational Testing Service (ETS). MEASURING THE POWER OF LEARNING is a trademark of ETS.

*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.

### I. Impacts of Computing

* 1. Understands and applies knowledge of impact
	 of, obstacles to, and effects of computing
1. Understand computing as a way of expressing creativity, solving problems, enabling communication, and fostering innovation in a variety of fields and careers
2. recognize that computers can be used to showcase creativity
3. recognize the benefits of using computers to solve problems
4. provide examples of how computers enable communication and collaboration
5. provide examples of how computers foster innovation
6. Know the obstacles to equal access to computing among different groups and
the impact of those obstacles
7. 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
8. identify factors that contribute to the digital divide
9. match obstacles to equal access with effective solutions
10. Understand beneficial and harmful effects of computing innovations and the trade-offs between them
11. analyze computing innovations in terms of their social, economic, and cultural impacts, both beneficial and harmful
12. 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
13. 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)
14. using correct vocabulary, describe
how different methods of protecting intellectual property rights work
15. given a context, identify appropriate methods of protecting intellectual property rights
16. identify and compare trade-offs between different methods of protecting intellectual property rights
17. Understand ethical and unethical computing practices and their social, economic, and cultural implications
18. identify ethical and unethical computing practices in context
19. describe the social, economic, and cultural implications of ethical and unethical computing practices
20. identify the conditions under which a given computing practice is ethical or legal
21. Know privacy and security issues regarding the acquisition, use, and disclosure of information in a digital world
22. using correct vocabulary, describe privacy and security issues
23. in context, identify appropriate strategies to safeguard privacy and ensure security
24. describe trade-offs between local and cloud-based data storage
25. identify methods that digital services use to collect information about users

### II. Algorithms and Computational Thinking

* 1. Understands and applies knowledge of abstraction, pattern recognition, problem
	decomposition, number base conversion,
	and algorithm formats
1. Understand abstraction as a foundation
of computer science
2. identify, create, or complete the
correct ordering, from low to high,
of an abstraction hierarchy
3. identify abstractions in context
4. identify details that can be removed from a solution in order to generalize it
5. Know how to use pattern recognition, problem decomposition, and abstraction
to develop an algorithm
6. given a table of values or other data source, identify the patterns in the data and identify algorithms that could produce the patterns
7. identify components that could be part of an algorithm to solve a problem
8. identify actions and actors when decomposing a problem
9. identify appropriate decomposition strategies
10. Understand number base conversion and binary, decimal, and hexadecimal number systems
11. convert between number bases
12. analyze and compare representations
of numbers in different bases
13. Understand how to develop and analyze algorithms expressed in multiple formats (e.g., natural language, flowcharts, pseudocode)
14. interpret diagrams that describe algorithms, given an explanation of the symbols used
15. compare algorithms written in multiple formats
16. trace and analyze algorithms written in different formats
17. 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
18. 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
19. identify and compare algorithms that
are linear, quadratic, exponential, or logarithmic
20. recognize the existence of problems that cannot be solved by a computer
21. in context, identify factors that prevent
a problem from being solvable
22. identify situations where heuristic solutions are useful
23. in context, identify space and time limitations of computational solutions
to problems
24. Understand searching and sorting algorithms; can analyze sorting algorithms for correctness and can analyze searching algorithms for correctness and efficiency
25. trace algorithms and predict output and intermediate results
26. calculate the number of comparisons required for linear and binary search algorithms
27. Understand simple recursive algorithms (e.g., *n* factorial, sum of first *n* integers)
28. trace simple recursive algorithms
29. provide missing steps in incomplete simple recursive algorithms
30. identify parts of a recursive algorithm (e.g., base or stopping condition, recursive call)
31. identify errors in simple recursive algorithms
32. identify an iterative algorithm that is equivalent to a recursive algorithm
33. Be familiar with the use of randomization
in computing
34. identify appropriate uses of randomization in a variety of applications
35. identify the difference between random and pseudorandom numbers

### III. Programming

* 1. Understands and applies knowledge of
	programming control structures, standard
	operators, variables, correctness, extensibility,
	modifiability, and reusability
1. Understand how to write and modify computer programs in a text-based programming language
2. describe what a program does or be able to choose the code segment that correctly implements a given intended purpose
3. identify missing code in a code segment with a stated intended purpose
4. place statements in appropriate order
to create a correct program
5. identify how changing one part of a code segment will affect the output
6. Understand how to analyze computer programs in terms of correctness
7. trace code and indicate the output printed or the value of variables after code segment execution
8. indicate the inputs that produce given outputs for a code segment
9. describe what a program does or choose the code segment that correctly implements a given intended purpose
10. identify valid preconditions and postconditions
11. compare two code segments or algorithms
12. identify the type of error produced by
a code segment (i.e., syntax, runtime, compile-time, overflow, round-off, logic)
13. identify errors in incorrect code and changes that can be made to correct them
14. Know the concepts of extensibility, modifiability, and reusability
15. identify the meaning of the terms
16. identify functionally equivalent statements or code segments that
differ in one of these three ways
17. identify situations where the use of constants or variables would be preferred over hard-coded values
18. identify opportunities for parameterization
19. choose code that improves on given code by making it more extensible, modifiable, or reusable
20. identify changes that would improve
a given code segment
21. Understand the three basic constructs used in programming: sequence, selection, and iteration
22. trace code and indicate the output printed or the value of variables after code segment execution
23. indicate inputs that produce given outputs for a code segment
24. describe what a program does or choose the code segment that correctly implements a given intended purpose
25. identify missing code in a code segment with a stated intended purpose
26. identify equivalent statements or code segments
27. identify the three constructs when used in code
28. identify which of the constructs are needed to implement given functionality
29. convert code that does not use iteration to equivalent code that uses iteration
30. Understand how to use standard operators (i.e., assignment, arithmetic, relational, logical) and operator precedence to write programs
31. trace code and indicate the output displayed or the value of variables after code segment execution
32. indicate inputs that produce given outputs for a code segment
33. describe what a program does or choose the code segment that correctly implements a stated intended purpose
34. identify missing code in a code segment with a stated intended purpose
35. identify equivalent statements or code segments
36. place statements in appropriate order to create a correct program
37. use Boolean algebra to identify equivalent Boolean expressions
38. write a Boolean expression equivalent to given code, or identify code equivalent to a given Boolean expression or English description
39. identify the correct implementation of a given formula, including formulas with fractions
40. evaluate expressions that include arithmetic operations
41. Understand how to use variables and a variety of data types
42. identify variables and data types (e.g., integers, floating point, string, Booleans, arrays/lists)
43. identify the need for type conversion
44. trace code and indicate the output printed or the value of variables after code segment execution
45. indicate the inputs that produce given outputs for a code segment
46. describe what a program does or choose the code segment that correctly implements a stated intended purpose
47. identify missing code in a code segment with a stated intended purpose
48. identify equivalent statements or code segments
49. place statements in appropriate order
to create a correct program
50. describe the difference between integer and floating point numeric data types
51. describe the difference between integer and floating point division
52. describe the benefits of the use of each data type
53. distinguish between global and local scope
54. identify the most appropriate data type in a given context
55. 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
56. Understand how to write and call procedures with parameters and return values
57. trace code and indicate the output printed or the value of variables after code segment execution
58. indicate inputs that produce given outputs for a code segment
59. describe what a program does or choose the code segment that correctly implements a stated intended purpose
60. identify missing code in a code segment with a stated intended purpose
61. identify equivalent statements or code segments
62. place statements in appropriate order
to create a correct program
63. trace code when references to objects and arrays are passed to procedures
64. trace code that includes nested procedure calls
65. Know the concepts of event-driven programs that respond to external events (e.g., sensors, messages, clicks)
66. trace code and indicate the output printed or the value of variables after code segment execution
67. indicate inputs that produce given outputs for a code segment
68. describe what a program does or choose the code segment that correctly implements a stated intended purpose
69. identify missing code in a code segment with a stated intended purpose
70. identify possible errors due to asynchronous events
71. identify aspects of concurrency in event-driven programming
72. Be familiar with usability and user experience (e.g., ease of use and accessibility)
73. identify code that improves on given code in terms of usability or user experience
74. identify meaningful error messages
75. identify features that improve accessibility
76. Be familiar with dictionaries/maps, stacks, and queues
77. identify a data structure based on a description of behavior or appropriate use
78. given goals, constraints, or context, identify the most appropriate data structure
79. trace code that uses a particular data structure
80. Understand how to use debugging techniques and appropriate test cases
81. identify which test cases are most useful for given code
82. differentiate between different types of errors (e.g., overflow, round-off, syntax, runtime, compile-time, logic)
83. describe useful debugging techniques (e.g., where to put print statements)
84. differentiate between empirical testing and proof
85. identify errors in code and solutions
to those errors
86. Be familiar with characteristics of well-documented computer programs that are usable, readable, and modular
87. identify characteristics of good documentation
88. identify good and poor documentation practices in context
89. Be familiar with techniques to obtain and use feedback to produce high-quality code (e.g., code reviews, peer feedback, end user feedback)
90. identify situations in which each of the three listed techniques are useful
91. Know how to use libraries and APIs
92. identify correct call(s) and use of return values given an API definition
93. identify reasons to use or not use libraries in place of writing original code
94. identify applications (e.g., math libraries, random number generation) that use APIs
95. Understand programming techniques to validate correct input and detect incorrect input
96. identify effective input data validation strategies
97. compare data validation (proper range and format) and data verification (e.g., password verification)
98. identify improvements to code for which data validation is required
99. Be familiar with the features and capabilities of integrated development environments (IDEs)
100. identify components of IDEs
101. identify benefits and drawbacks of using IDEs
102. identify the costs and benefits of context editors
103. Be familiar with the differences between low- and high-level programming languages
104. identify characteristics of low- and high-level languages
105. Be familiar with different programming paradigms
106. identify the terminology of procedural programming
107. identify the terminology of object-oriented programming
108. compare programming paradigms
109. Know object-oriented programming concepts
110. identify classes, instance variables,
and methods given a diagram
111. identify the benefits of inheritance
and encapsulation
112. identify distinctions between overloading and overriding
113. Be familiar with program compilation and program interpretation
114. identify differences between compilation and interpretation
115. identify differences between source code and object code

### IV. Data

* 1. Understands and applies knowledge of
	digitalization, data encryption and decryption,
	and computational tools
1. Understand bits as the universal medium for expressing digital information
2. perform calculations, using bits and bytes
3. determine the number of bits and bytes required to store a given amount of data
4. given the description of an encoding scheme, encode or decode data
5. describe lossy and lossless data compression
6. explain why binary numbers are fundamental to the operation of computer systems
7. Be familiar with concepts of data encryption and decryption
8. distinguish between encoding and encryption
9. identify trade-offs in the use of data encryption
10. Know how to use computational tools, including spreadsheets, to analyze data
in order to discover, explain, and visualize patterns, connections, and trends
11. transform data to make it more useful
12. identify specific data or characteristics of specific data that need to be removed or modified before an entire data set can be used
13. 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
14. Be familiar with the use of computing
in simulation and modeling
15. 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
16. trace code in a simulation context
17. identify missing code in a simulation context
18. identify the impact of changes to simulations (e.g., more or fewer variables, more or less data)
19. identify applications of simulation
and modeling
20. Be familiar with methods to store, manage, and manipulate data
21. use terminology and concepts of files and databases
22. identify measures of file size
(e.g., byte, kilo, mega, giga, tera, peta)
23. identify issues connected with the storage requirements of computing applications, including scale, redundancy, and backup
24. Be familiar with a variety of computational methods for data collection, aggregation, and generation
25. identify the benefits of working with publicly available data sets
26. identify the types of data generated by surveys and sensors
27. identify examples of crowdsourcing and citizen science
28. identify appropriate data-collection methods for a given context and purpose

### V. Computing Systems and Networks

* 1. Understands and applies knowledge
	of operating systems, computing systems,
	communication between devices, and cloud computing
1. Know that operating systems are programs that control and coordinate interactions between hardware and software components
2. identify hardware components and their functions
3. identify software components and their functions
4. identify common operating systems tasks
5. identify resource issues that have an impact on functionality
6. Be familiar with computing systems embedded in everyday objects (e.g., Internet of Things [IoT], ATMs, medical devices)
7. describe what an embedded system is
8. define what the IoT is and how it is used
9. describe how sensors are used in embedded systems
10. Know the capabilities, features, and uses
of different types of computing systems (e.g., desktop, mobile, cluster)
11. identify capabilities, features, and uses for each type of computer system
12. identify criteria to evaluate and compare computing systems
13. Be familiar with computers as layers of abstraction from hardware (e.g., logic gates, chips) to software (e.g., system software, applications)
14. identify appropriate abstraction layers for hardware and software components
15. Be familiar with the steps required to execute a computer program (fetch-decode-execute cycles)
16. describe what happens during fetch, decode, and execute, including the order of the steps in the cycle
17. Be familiar with trade-offs between local, network, and cloud computing and storage
18. identify advantages and disadvantages in terms of performance, cost, security, reliability, and collaboration
19. identify means of storing binary data
20. Be familiar with communication between devices
21. identify and compare wireless communication systems
22. identify and compare wired communication systems
23. identify and compare network types
	1. Understands and applies knowledge of
	 networks, including security issues and
	 the Web
24. Know components of networks
25. identify network hardware devices
and their functions
26. describe possible abstraction models
of networks
27. Be familiar with factors that have an impact on network functionality
28. define basic terminology
(e.g., bandwidth, load, latency)
29. estimate necessary bandwidth and
data size for a given situation
30. identify critical resources for a given situation
31. Be familiar with how Internet and Web protocols work
32. describe the purpose of protocols and identify common Internet and Web protocols
33. compare IPv4 and IPv6
34. identify and describe the basic parts
of a URL (e.g., protocol, subdomain, domain name, port, path)
35. describe the hierarchical structure of names in the domain name system (DNS)
36. describe the purpose and function of
IP addressing
37. identify how Internet protocols address reliability, redundancy, and error handling
38. Be familiar with digital and physical strategies for maintaining security
39. identify characteristics of strong passwords (e.g., length, bits per character)
40. identify digital and physical security strategies
41. identify trade-offs in the use of security measures (e.g., encryption, decryption, digital signatures and certificates)
42. Be familiar with concepts of cybersecurity
43. identify and define the five pillars of cybersecurity: confidentiality, integrity, availability, nonrepudiation, and authentication
44. Be familiar with the components that make up the Web (e.g., HTTP, HTML, browsers, servers, clients)
45. identify the uses of markup languages
46. 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 \*/ |
| PrintA 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[]** aleft arrow{1, 2, 3}**int** b[0..2]left arrow{1, 2, 3}**int[][]** ca[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** |