Readings from Programming with Python 1st edition Mcmullen Solutions Manual

SKU: 9780357637456-SOLUTIONS

Instructor Manual

McMullen, Matthews, and Parsons, Programming with Python ©2023, 978-035-763-7456; Module 1: Computational Thinking

Table of Contents

Purpose and Perspective of the Chapter

The purpose of this module is to introduce algorithms to students. Students will learn steps and terminology related to creating an algorithm and apply this knowledge to the programming environment. Programming algorithms are specifically reviewed with an emphasis on how to assess a problem and decompose the processes into smaller subcomponents or modules. The importance of decomposition as an important tool for computer scientists is emphasized. Specifically, three types of decomposition are covered in more detail: structural, functional, and object-oriented decomposition. The importance of algorithm efficiency is presented along with a list of characteristics an effective algorithm possesses. Students will also learn how to recognize the various types of patterns frequently found in an algorithm: fill-in-the-blank, repetitive, and classification patterns. The module concludes with a discussion of abstraction. Students will learn why abstraction is an important computer science concept.

Cengage Supplements

The following product-level supplements provide additional information that may help you in preparing your course. They are available in the Instructor Resource Center.

- Test Bank: Cengage Testing, powered by Cognero® (contains assessment questions and problems)
- PowerPoint (provides text-based lectures and presentations)
- Complete List of Learning Objectives (provides a comprehensive list of learning objectives for all modules)
- Python Code Files (includes code files of figures from the modules and fully executable Python programs)
- MindTap Educator Guide (contains a detailed outline of the MindTap)
- Guide to Teaching Online (includes pedagogical considerations and resources for teaching online)

List of Student Downloads

Students should download the following items from the Student Companion Center to complete the activities and assignments related to this chapter:

- Python Snippets ReadMe (provides helpful information related to the embedded coding Snippets)

Module Objectives

The following objectives are addressed in this chapter:

1.1. Algorithms

1.1.1 Define the term “algorithm” as a series of steps for solving a problem or carrying out a task.

1.1.2 State that algorithms are the underlying logic for computer programs.

1.1.3 Define the term “computer program.”

1.1.4 Provide examples of algorithms used in everyday technology applications.

1.1.5 Confirm that there can be more than one algorithm for a task or problem and that some algorithms may be more efficient than others.

1.1.6 Explain why computer scientists are interested in algorithm efficiency.

1.1.7 List the characteristics of an effective algorithm.

1.1.8 Write an algorithm for accomplishing a simple, everyday technology task.

1.1.9 Write an alternate algorithm for an everyday technology task.

1.1.10 Select the more efficient of the two algorithms you have written.

1.2. Decomposition

1.2.1 Define the term “decomposition” as a technique for dividing a complex problem or solution into smaller parts.

1.2.2 Explain why decomposition is an important tool for computer scientists.

1.2.3 Differentiate the concepts of algorithms and decomposition.

1.2.4 Identify examples of structural decomposition.

1.2.5 Identify examples of functional decomposition.

1.2.6 Identify examples of object-oriented decomposition.

1.2.7 Provide examples of decomposition in technology applications.

1.2.8 Explain how dependencies and cohesion relate to decomposition.

1.3. Pattern Identification

1.3.1 Define the term “pattern identification” as a technique for recognizing similarities or characteristics among the elements of a task or problem.

1.3.2 Identify examples of fill-in-the-blank patterns.

1.3.3 Identify examples of repetitive patterns.

1.3.4 Identify examples of classification patterns.

1.3.5 Provide examples of pattern identification in the real world and in technology applications.

1.4. Abstraction

1.4.1 Define the term "abstraction" as a technique for generalization and for simplifying levels of complexity.

1.4.2 Explain why abstraction is an important computer science concept.

1.4.3 Provide an example illustrating how abstraction can help identify variables.

1.4.4 Provide examples of technology applications that have abstracted or hidden details.

1.4.5 Provide an example illustrating the use of a class as an abstraction of a set of objects.

1.4.6 Explain how the black box concept is an implementation of abstraction.

1.4.7 Identify appropriate levels of abstraction.

Complete List of Chapter Activities and Assessments

For additional guidance refer to the Teaching Online Guide.

Chapter Objective	PPT slide	Activity/Assessment	Activity Duration
1.1.1, 1.1.5, 1.1.7 to 1.1.10	10	Activity 1.1: Breakout Groups: Traveler Algorithm	30-40 minutes
1.1.2, 1.2.1, 1.2.4 to 1.2.6	17-18	Activity 1.2: Knowledge Check	5-10 minutes
1.1.4, 1.1.7 to 1.1.9	21	Activity 1.3: Discussion	10-15 minutes
1.3.1, 1.4.1, 1.4.6	26-27	Activity 1.4: Knowledge Check	5-10 minutes
1.2.4, 1.2.5, 1.2.6, 1.3.5, 1.4.6	30	Self-Assessment	10-15 minutes

[return to top]

Key Terms

abstraction: a representation that hides details, simplifies complexity, substitutes a generalization for something specific, and allows an algorithm to work for multiple inputs.
algorithm: a series of steps for solving a problem or carrying out a task.
attributes: nouns that relate to the characteristics of an object.
classes: templates that specify the attributes for a classification of objects.
classification patterns: patterns found in the attributes that describe any person or object.
computational thinking: a set of techniques designed to formulate problems and their solutions.
computer program: a set of instructions, written in a programming language such as Python, that performs a specific task when executed by a digital device.
decomposition: the process of dividing an algorithm for a complex app into smaller parts so that the algorithm is easier to formulate.
functional decomposition: a process that breaks down modules into smaller actions, processes, or steps.
level of abstraction: the level of detail that is hidden by a representation.
methods: actions that an object can perform.
modules: structural units that perform distinct tasks.
objects: programming constructs that represent people, places, or things.
object-oriented decomposition: a way to apply decomposition to a module by looking for logical and physical objects that a computer program will manipulate.
pattern identification: the process of finding similarities in procedures and tasks.
programming algorithm: a set of steps that specifies the underlying logic and structure for the statements in a computer program.
structural decomposition: a process that identifies a hierarchy of building-block units for a program concept.

[return to top]

[return to top]

Module Outline

In the outline below, each element includes references (in parentheses) to related content. "MOD.##" refers to the module objective; "PPT Slide #" refers to the slide number in the PowerPoint deck for this chapter (provided in the PowerPoints section of the Instructor Resource Center); and, as applicable for each discipline, accreditation or certification standards ("BL 1.3.3"). Introduce the module and use the Ice Breaker in the PPT if desired, and if one is provided for this module. Review learning objectives for Module 1 (PPT Slides 3-6).

1.1 Algorithms (1.1.1 to 1.1.10, PPT slides 7-9)

I. Algorithm Basics (1.1.1, 1.1.4)

a. Remind students that passwords can be used to protect online accounts and review the basic process.
b. Introduce the concept of two-factor authentication as an extra layer of account protection from perpetrators with bad intentions.
c. Use Figure 1-1 to illustrate a common form of two-factor authentication that sends a personal identification number (PIN) to a user's cell phone and walk through the steps of this process.
d. Define an **algorithm** as a series of steps for solving a problem or carrying out a task. Note that the procedure for two-factor authentication can be considered an example of an algorithm.
e. Mention that algorithms exist for everyday tasks and tasks that involve technology. Review the following example tasks:
   - A recipe for baking brownies
   - The steps for changing a tire
   - The instructions for pairing a smart watch with a phone
   - The payment process at an online store
   - The procedure for posting a tweet
f. Take 5 minutes to help students gain experience writing a short algorithm. Break the class into small groups of students. Select two of the task examples above (or select a new idea). Assign each of the selected tasks to at least two student groups. Ask each student group to devise a short algorithm to complete their assigned task. Have each student group present their solution to the class for a short discussion. Those groups that completed the same task can compare their results.

II. Programming Algorithms (1.1.2, 1.1.3, 1.1.5)

a. Remind students that in general, an algorithm is a series of steps for solving a problem or carrying out a task. Note that algorithms are an important programming tool.
b. Define a **programming algorithm** as a set of steps that specifies the underlying logic and structure for the statements in a computer program. Explain that programming algorithms can be considered as the blueprints for computer programs.
c. Define a **computer program** as a set of instructions, written in a programming language such as Python, that performs a specific task when executed by a digital device. Mention that a computer program is an implementation of an algorithm.

d. To further explain a programming algorithm, refer to the first question and answer set that illustrates two programming algorithms. Note that because Algorithm 2 provides instructions that should be for the computer and not the user, it represents a programming algorithm.
e. To illustrate algorithm efficiency, refer to the second question and answer set. Explain why Algorithm 2 in this example is more efficient.

III. "Good" Algorithms (1.1.6, 1.1.7)

a. Mention that a good algorithm tends to produce a computer program that operates efficiently, quickly, and reliably.
b. Spend time reviewing characteristics of good algorithms.
   - Input: The algorithm applies to a set of specified inputs.
   - Output: The algorithm produces one or more outputs.
   - Finite: The algorithm terminates after a finite number of steps.
   - Precise: Each step of the algorithm is clear and unambiguous.
   - Effective: The algorithm successfully produces the correct output.
c. Use Figure 1-2 to explain how a programmer might easily check to make sure an algorithm satisfies all the criteria for a good algorithm.

IV. Selecting and Creating Algorithms (1.1.8, 1.1.9, 1.1.10)

a. Mention that before coding, programmers consider various algorithms that might apply to a problem. Review the three ways a programmer might come up with an algorithm.
   - Use a standard algorithm.
   - Perform the task manually.
   - Apply computational thinking techniques.
b. Explain that when programmers use **computational thinking**, they employ a set of techniques designed to formulate problems and their solutions.
c. Note that programmers can use computational thinking techniques such as decomposition, pattern identification, and abstraction to devise efficient algorithms.

1.2 Decomposition (1.2.1 to 1.2.8, PPT Slides 11-16, 19)

I. Decomposition Basics (1.2.1)

a. Reference Figure 1-3 to illustrate how an app can have many components. Note that all of these components together can lead to an extensive algorithm.

b. Define **decomposition** as the process of dividing the algorithm for an extensive app into smaller parts so that the algorithm is easier to formulate. Discuss how decomposition can help a programmer deal with smaller, more manageable pieces of a problem. Be sure to emphasize that a programmer will decompose a problem to make it easier to solve.

II. Structural Decomposition (1.2.2, 1.2.3, 1.2.4, 1.2.7)

a. Point out that the first step in decomposition is to identify structural units that perform distinct tasks.
b. Use Figure 1-4 to illustrate how a mobile banking app might be broken into structural units, called **modules**.
c. Define **structural decomposition** as a process that identifies a hierarchy of structural units. Explain that at the lowest levels of the hierarchy are modules that have a manageable scope for creating algorithms. Point out that in Figure 1-4 the lowest levels are yellow.
d. To further explain structural decomposition, refer to the question and answer set.
e. Take the time to review the provided tips for creating a structural decomposition diagram.

III. Functional Decomposition (1.2.5)

a. Explain that **functional decomposition** breaks down modules into smaller actions, processes, or steps.
b. Use Figure 1-5 to illustrate a functional decomposition of the two-factor authentication module in an online banking app. Point out that the levels of the functional decomposition diagram get more specific until the nodes in the lowest levels begin to reveal instructions that should be incorporated into an algorithm.
c. Take the time to review the provided tips for constructing functional decomposition diagrams and deriving algorithms from them.

IV. Object-Oriented Decomposition (1.2.6)

a. Introduce the concept of **object-oriented decomposition** by explaining that it applies decomposition to a module by looking for logical and physical objects that a computer program will manipulate.
b. Use Figure 1-6 to illustrate an object-oriented decomposition of the two-factor authentication module for the mobile banking app.

c. Explain that an object-oriented decomposition does not produce a hierarchy. Instead, it produces a collection of **objects** that can represent people, places, or things and have associated **attributes** and **methods**.
d. Take the time to review the provided tips for object-oriented decomposition.

V. Dependencies and Cohesion (1.2.8)

a. Introduce this topic by explaining that in practice, there may be several viable ways to apply decomposition. Point out that an effective breakdown minimizes dependencies and maximizes cohesion among the various parts.
b. Review the two principles of decomposition:
   - Minimize dependencies. Although input and output may flow between nodes, changing the instructions in one module or object should not require changes to others.
   - Maximize cohesion. Each object or module contains attributes, methods, or instructions that perform a single logical task or represent a single entity.

1.3 Pattern Identification (1.3.1 to 1.3.5, PPT Slide 20)

I. Pattern Identification Basics (1.3.1, 1.3.2)

a. Introduce the Amaze-Your-Friends math trick, explaining that it is used to simply and quickly calculate the sum of numbers.
b. Ask each student to work through the four steps in the Amaze-Your-Friends math trick to quickly calculate the sum of the numbers from 1 to 10. Verify students calculated 55 as the answer.
c. Refer to the question and answer set to challenge your students even more. Ask students to work through the four steps in the Amaze-Your-Friends math trick to quickly calculate the sum of the number from 1 to 200. Verify students multiply 100 x 201 to calculate a result of 20,100.
d. Emphasize that there is a pattern allowing the Amaze-Your-Friends math trick to work successfully—in this case, a fill-in-the-blank pattern.
e. Define **pattern identification** as the process of finding similarities in procedures and tasks.
f. Point out that pattern identification is a useful computational thinking technique for creating algorithms that can be used and reused on different data sets.
g. Explain that by recognizing the pattern in the Amaze-Your-Friends math trick, the students can use the algorithm to find the total of any series of numbers.

II. Repetitive Patterns (1.3.3)

a. Explain that repetitive patterns can also be used as a programmer analyzes tasks and solves problems.
b. Review the ten-step algorithm that describes how to handle logins to a social media site. Read each line aloud, emphasizing the repetitive instructions.
c. Use the question and answer set to highlight the number of times the repetitive instructions occur.
d. Explain how the algorithm can be streamlined by using a repeat statement. Review the updated, streamlined algorithm in the answer section.

III. Classification Patterns (1.3.4, 1.3.5)

a. Explain that programmers can often discover **classification patterns** in the attributes that describe any person or object. Emphasize that identifying classification patterns can help a programmer design programs that involve databases because the template identifies fields, such as User ID, that contain data.
b. To illustrate classification patterns, review the login credential sets for two fictitious users, Lee and Priya, who subscribe to a social media site.
c. Point out that each user has three attributes: a user ID, a password, and a mobile number. Emphasize that by recognizing a pattern like this, a programmer can create a template for any user's login credentials. Review the example template provided.
d. Explain that classification patterns are useful if a programmer wants to design programs based on the interactions among a variety of objects, rather than a step-by-step algorithm. Note that in some programming circles, templates are called **classes** because they specify the attributes for a classification of objects.
e. Provide examples of classification patterns that are based on the interactions among a variety of objects.
    - People classified as social media subscribers have attributes for login credentials.
    - Vehicles classified as cars have attributes such as color, make, model, and VIN number.
    - Businesses classified as restaurants have a name, hours of operation, and a menu.

1.4 Abstraction (1.4.1 to 1.4.7, PPT slides 22-25, 28-29)

I. Abstraction Basics (1.4.1, 1.4.2, 1.4.3)

a. Remind students of the Amaze-Your-Friends math trick, noting that by identifying a pattern, they were able to formulate a general algorithm that works for a sequence of any length.
b. Review the fill-in-the-blank pattern for the Amaze-Your-Friends math trick algorithm, emphasizing that the blank lines are an abstraction that represents the last number in the sequence.
c. Explain that an **abstraction** hides details, simplifies complexity, substitutes a generalization for something specific, and allows an algorithm to work for multiple inputs.
d. Emphasize that abstraction is a key element of computational thinking and helps programmers in a multitude of ways.
e. Ask students to raise their hands if they have programmed before. Mention that they may recognize that in the Amaze-Your-Friends algorithm, the blanks could become a variable with a name such as `last_number`. Point out that `result` and `sum` are also variables because they represent values that change depending on the numbers in the sequence.
f. Conclude this section by noting that a variable is an abstraction because rather than representing a specific number, it can be used to represent many different numbers.

II. Classes and Objects (1.4.4, 1.4.5)

a. Introduce this topic by pointing out that abstraction has uses other than identifying variables.
b. Discuss why abstraction is important for understanding how to represent real-world and conceptual objects.
c. Have students recall the pattern discovered for social media login credentials. Explain that with a little modification, the pattern becomes a template that can be applied to any subscriber.
   - Class: LoginCredentials
   - Attribute: user_ID
   - Attribute: user_password
   - Attribute: mobile_number
d. Mention that the LoginCredentials class is an abstraction that contains a set of attributes. Explain that the class was formed by abstracting away, or removing, details for any specific subscriber.
e. Note that abstractions are handy for any programs that deal with real-world objects in addition to technology objects, such as login credentials.
f. Refer to the question and answer set to encourage students to envision a class that is an abstraction of the collection of objects shown in Figure 1-7. Explain that the glassware class could have these attributes:

- Class: Glassware
- Attribute: Color
- Attribute: Capacity
- Attribute: Style

III. Black Boxes (1.4.6)

a. Use Figure 1-8 to illustrate abstraction applied to driving a car. Explain that to drive a car, a driver does not have to know exactly what goes on under the hood. Point out that the engine is essentially a "black box" that the student controls using a few simple inputs such as the gas pedal, brake, and steering wheel. The details of the engine are hidden. Mention that in computer science terminology, these details have been "abstracted away."
b. Use Figure 1-9 to illustrate that in concept, a black box is anything that accepts some type of input and performs a specific process to produce output without requiring an understanding of its internal workings.
c. Explain why the black-box concept of abstraction is a fundamental aspect of computer science. Emphasize that this is possible because computer programs are abstractions.
d. Using Figure 1-10, explain that programmers make extensive use of abstraction within programs by creating a set of instructions that functions like a black box.
e. Note that programming languages often have built-in abstractions that perform standard tasks. Use the example of a built-in random function that generates a random number when given a range, such as 1–100. Mention that programmers can incorporate the random function in a program without knowing how it works internally.

IV. Levels of Abstraction (1.4.7)

a. Encourage students by indicating that applying the correct level of abstraction to their programs may take a little practice.
b. Note that a **level of abstraction** relates to the amount of detail that is hidden.
c. Use Figure 1-11 to emphasize that abstracting out too much detail can make a program too generalized, whereas neglecting abstraction can produce programs that are too specific to work with a wide variety of data.

[return to top]

Discussion Questions

You can assign these questions several ways: in a discussion forum in your LMS; as whole-class discussions in person; or as a partner or group activity in class.

1. Activity 1.3 Discussion (PPT Slide 21) Duration 10-15 minutes.
    a. Question 1: Discuss some algorithms that are used by your favorite games and apps. (1.1.4)
        a. **Answer:** The module mentions the processes for paying for something purchased through a website or app and for posting text to social media as examples of algorithms used in technology. Other representative examples include procedures for determining the result of one character's attack on another character in a video game, providing directions to a desired location in a maps app, and applying a filter to a store's product catalog to display only certain products of interest to the user in their app.
    b. Question 2: What is the best way to ensure that an algorithm is free of logical errors? (1.1.7 to 1.1.9)
        a. **Answer:** An algorithm that is free of logical errors is effective (successfully produces the correct output for each given input). Therefore, a reasonable way to detect and fix logical errors would be to perform robust testing of the algorithm (or code created from it) to see if it reliably produces the desired output. Carefully checking that each step of the algorithm in fact accomplishes what it is supposed to before the procedure moves to the next step is another.

[return to top]

Suggested Usage for Lab Activities

1. No Lab Activities for Module 1.

[return to top]

Additional Activities and Assignments

1. **Icebreaker (PPT slide 2)**
    a. The class will be broken up into pairs of students. (Can also be done using Breakout Rooms in Zoom.)
    b. Each pair of students will interview each other to discover one of the following two things:

   i. Favorite video/computer game of all time
   ii. Earliest computer/electronic device memory
c. Then each pair will introduce each other to the class.

2. **Activity 1.1: Breakout Groups Activity: Travel Algorithm (1.1.1, 1.1.5, 1.1.7 to 1.1.10; PPT slide 10)**
   a. Form small groups based on how you commute to school: walk, ride a bike, drive a car, drive a motorcycle, take a bus, take a train, or use multiple modes of transportation.
   b. Work together to write an algorithm for your mode of transportation (this should take 5-10 minutes).
   c. Pass your algorithm to the group of students next to yours in a clockwise direction.
   d. Evaluate and critique the algorithm your group received (this should take 5-10 minutes).
   e. Take turns with the other groups to present your algorithm critique to the class so that everyone can learn from each other.
3. **Activities 1.2 and 1.4: Knowledge Check (1.1.2, 1.2.1, 1.2.4 to 1.2.6, 1.3.1, 1.4.1, 1.4.6; PPT slides 17-18, 26-27)**
   a. A computer program is an implementation of a(n) _____ in a programming language.
   Answer: algorithm (1.1.2)
   b. Among the strategies for dividing a software application into smaller parts are _____, which breaks components into smaller actions, processes, or steps; _____, which identifies a hierarchy of building-block units; and _____, which identifies a collection of nodes along with their characteristics and actions.
   Answer: functional decomposition; structural decomposition; object-oriented decomposition (1.2.1, 1.2.4 to 1.2.6)
   c. When you look for similarities in two procedures so that you can rewrite the procedures to share a common set of instructions, you are performing _____.
   Answer: pattern identification (1.3.1)
   d. When you substitute a descriptive name for a value that can vary within a set of instructions, you are creating a(n) _____ by substituting a generalization for something specific.
   Answer: abstraction (1.4.1)
   e. Anything that accepts some type of input and performs a specific process to produce output without requiring an understanding of its internal workings can be termed a(n) _____.
   Answer: black box (1.4.6)

[return to top]

Additional Resources

Cengage Video Resources

- MindTap Videos:
  - What is Computer Programming? 1:19 min.
  - Top-Down Design 4:35 min.
  - Methods 2:32 min.

Internet Resources

- **BBC Bitesize: Computational thinking (webpage)** (1.1.1 to 1.1.10, 1.2.1 to 1.2.3, 1.2.7, 1.3.1, 1.3.5, 1.4.1 to 1.4.4)
- **Geeks for Geeks: Introduction to Algorithms (webpage)** (1.1.1 to 1.1.10)
- **The Conversation: Algorithms are everywhere but what will it take for us to trust them? (webpage)** (1.1.4 to 1.1.7)

[return to top]

Appendix

Generic Rubrics

Providing students with rubrics helps them understand expectations and components of assignments. Rubrics help students become more aware of their learning process and progress, and they improve students' work through timely and detailed feedback. Customize these rubrics as you wish.

Standard Discussion Rubric

Criteria	Meets Requirements	Needs Improvement	Incomplete
Participation	Submits or participates in discussion by the posted deadlines. Follows all assignment. instructions for initial post and responses. 5 points	Does not participate or submit discussion by the posted deadlines. Does not follow instructions for initial post and responses. 3 points	Does not participate in discussion. 0 points
Contribution Quality	Comments stay on task. Comments add value to discussion topic. Comments motivate other students to respond. 20 points	Comments may not stay on task. Comments may not add value to discussion topic. Comments may not motivate other students to respond. 10 points	Does not participate in discussion. 0 points
Etiquette	Maintains appropriate language. Offers criticism in a constructive manner. Provides both positive and negative feedback. 5 points	Does not always maintain appropriate language. Offers criticism in an offensive manner. Provides only negative feedback. 3 points	Does not participate in discussion. 0 points

Standard Collaboration Rubric

Criteria	Meets Requirements	Needs Improvement	Incomplete
Share knowledge	Consistently and actively contributes knowledge, opinions, and skills. 5 points	Contributes to the group only when prompted. 3 points	Does not contribute to the group. 0 points
Adjust to unforeseen circumstances	Seeks different solutions, approaches, and strategies in an effective, original, and/or creative way. 5 points	Seeks a single solution, approach, or strategy, but does not pursue better or original alternatives. 3 points	Relies on the first solution generated and does not offer other solutions. 0 points

Make decisions	Helps the group identify necessary changes and encourages group action for change. 10 points	Participates in needed changes only when prompted. 5 points	Does not participate in decision making. 0 points
Build consensus	Values, encourages, and acknowledges the work of other group members. Takes responsibility for the assignment that reflects minority and majority conclusions of the group. 10 points	Listens attentively to members of the group but contributes little to the group and assignment. 5 points	Does not participate in building consensus within the group. 0 points
Complete deliverables (only if applicable)	Completes all assigned components or deliverables with quality, creativity, and accuracy. Turns in all assigned components by agreed upon deadlines without reminders. 10 points	Completes all assigned components with bare minimum of quality and creativity. Requires reminders and follow-up from team members in order to complete deliverables. 5 points	Does not complete assigned components or deliverables or turns them in late. Assigned components are poorly executed, inaccurate, or not functioning. Does not respond to communication from team. 0 points

Programming Rubrics

Providing students with programming rubrics helps them understand expectations and components of writing code. Rubrics help students become more aware of their learning process and progress, and they improve students' work through timely and detailed feedback. Customize these rubrics as you wish.

Standard Programming Rubric (Simple)

Criteria	Excellent	Good	Fair	Poor	None
Function: The code works without errors and fulfills all project requirements. Test cases for in- and out-of-range inputs are successful. Error handling is used correctly, and where necessary and appropriate.	4	3	2	1	0

Formatting: The code is readable, formatted properly with indents, and line spacing. Variables are named appropriately and consistently. The code is well-commented.	4	3	2	1	0
Simplicity: The code efficiently fulfills the project requirements. Hardcoded values are minimized. Complex programming logic is distributed in short, single-purpose lines of code.	4	3	2	1	0
Modularity: Programming follows the Don't Repeat Yourself best practice. Each method/function performs one discrete task. Variables are strongly typed and used for singular, discrete purposes.	4	3	2	1	0
Comprehension: The code demonstrates the student understands the concepts required. The student can explain how the program works. The student can explain why alternative solutions are not as good as the one(s) used.	4	3	2	1	0

Standard Programming Rubric (Detailed)

Criteria	4	3	2	1	0	Total
Function	The code works without errors and fulfills all project requirements. Test cases for in- and out-of-range inputs are successful. Error handling is used correctly, where necessary and appropriate.	The code works but may generate some errors with edge cases. Most project requirements are met. Test cases for in-range inputs are successful. Error handling, if required, is present.	The code compiles/executes but does not fulfill all project requirements. Test case for at least one in-range input is successful. Error handling, if required, is present.	The code is mostly complete but does not compile/execute. Error handling, if required, is present.	The code is substantially incomplete or nonfunctional. No error handling is present.	

Formatting	The code is readable and formatted properly into control structures with indents and line spacing. Variables are named appropriately and consistently. The code is well-commented.	The code is readable with appropriate line spacing but not indented. Variables are named appropriately but not consistently. The code is mostly commented.	The code is readable but written in large blocks without indents or spacing. Variables are inconsistently named. The code is commented sporadically.	The code is written haphazardly and/or appears to be copied from external sources with minimal modification. Variables are randomly named or single characters. Minimal comments are present.	No attempt made to format the code. Variables are single characters. No comments are present.	
Simplicity	The code efficiently accomplishes what is in the project requirements. Hardcoded values are used only where appropriate. Complex programming logic is distributed in short, single-purpose lines of code.	The code includes some unnecessary extra steps but accomplishes the project requirements. Some hardcoded values that will require future code updates are present. Some multifunction or multistep lines of code are used.	Numerous extra blocks of code and/or unused variables are present. The code includes many extra unnecessary steps. Hardcoded values are used where variables should be. Lines of code combine numerous logical functions/steps.	Code complexity is great enough to challenge future modification or debugging. Extra/unnecessary steps in code throughout. Programming flow is difficult to follow. Hardcoded values are used instead of variables.	Code complexity is too great to allow future modifications or debugging. Programming flow does not exist. No logical order to programmatic steps. Variables are not used or used incorrectly.	

Modularity	The code follows the Don't Repeat Yourself best practice. Each method/function performs one discrete task. Variables are strongly typed and used for singular, discrete purposes.	The code mostly follows the Don't Repeat Yourself best practice. Some methods/functions perform more than one discrete task. Occasional variable repurposing or reuse exists.	Logical blocks of code are duplicated in different methods/functions. Methods/functions combine multiple logical operations and steps. Variables are untyped and/or occasionally reused.	Minimal effort to create logical blocks of code. Procedural coding practices used for logical flow without code reuse. Variables are untyped and/or repurposed.	No attempt made to create modular code. All code exists in one large block. Variables, if present, are untyped and/or repurposed.	
Comprehension	The code clearly demonstrates the student understands the concepts required. The student can explain how the entire program works. The student can explain why alternative solutions are not as good as the one(s) used.	The code shows the student understands most of the concepts required. The student can explain how most of the program works. The student can explain at least one alternative solution.	The code shows the student is aware of some of the concepts required. The student can explain how some parts of the program work. The student can discuss alternative solutions.	The code shows the student is aware of at least one of the concepts required. The student can explain how at least one part of the program works. The student can discuss at least one alternative solution.	The code does not show an awareness of the concepts required. The student cannot explain how the program should work. The student cannot discuss alternative solutions.	
					Total	