- Another step is to use well-named variables and procedures. If they are descriptive names that communicate what they do, it is easier to trace through a program and understand what it is supposed to do. See Big Idea 3 for more information on variables. For example: *amtDue* is much clearer as a variable name than *ad*.
- Hand-tracing the flow of the program line-by-line is also an effective debugging tool. This is often the best way to find a logic error. You write down the variable names and values as they change along with iteration numbers of loops and check the code line-by-line.
- Many IDEs (integrated development environments) where you can write and test your code include coding visualizations. These show the code structure and each line that executes to help track where errors are occurring.
- Debuggers are programs written to test and debug other programs. Many IDEs include debuggers. These are like automated versions of hand-tracing and show your code step by step as it executes.
- The use of temporary print or DISPLAY statements inside a program help with debugging. By printing out intermediate values to the screen or on paper, or to indicate when a program reaches a certain section of code, the programmer can more easily identify the error. These extra print statements are removed after the error is resolved.

There are several types of errors that can occur when testing. The types you must know for this course are described below.

Types of Errors

Syntax errors deal with things like punctuation or missing parentheses ( ) and typos. These are identified when you try to compile your program and must be corrected before the program will run. Each programming language has its own syntax, or rules, that must be followed.

Runtime errors do not cause an error at compile time but occur when the program is executing. A common example is when a variable has a value of zero and the program tries to divide by it. The program may run successfully many times when the value is not zero, but will crash when it is zero.

Logic errors occur when the program produces unexpected results. These errors are typically harder to identify. You may think your program ran correctly, because there were no identified errors. The program can run, but the actual results would be different than expected.

The test cases with expected results can really help identify these. If you have a table of expected test results, and the program does not produce the expected value, you need to carefully check your code to find and correct the problem and be sure you understand what the program is supposed to do. The problem could be as simple as using "+" rather than "–" in a calculation, or you may have used the wrong variable name in a program statement.

Overflow errors occur in computers when an integer is larger than the programming language can hold. A fixed number of bits are assigned to hold integers in many programming languages. When the limit is reached, an overflow error occurs. This varies for different computers, and you are not expected to know this value for each computer for the AP exam. See the Big Idea 2 chapter about data for more information about overflow errors.

Reflection

There are two development processes you need to know for this course. Each involves reflection of results before moving on in the development process.

An *iterative* development process is a repetitive one. The steps are repeated several times throughout the project as new information is gathered or clarified, testing is performed and revisions are needed, or customers change their mind about what they need. Each iteration produces a better result. After working through the investigation and design phases, often the coding/prototyping, testing, and reflection steps are repeated multiple times until

the code works. Sometimes you do have to go back to other phases to work through those steps again. Feedback is a significant step in this process. You can create a working program, but if it is difficult to use or confusing, then people will not use it. Ask them! Ask a variety of people. You can ask your friends, but make sure to ask others as well. Friends may think too similarly to you or not want to tell you the truth.

An *incremental* development process breaks the pieces of the program into smaller sections. A section is coded, tested, modified, tested, and so on until it is working. Then work begins on another section. Once it is working, it is also tested for any interactions with sections of code already completed. Then work begins on the next module, and the cycle continues. The overall software system is built piece by piece, or incrementally, until completed. For example, if you are writing an app for a game, you might break it into the following sections:

- Instructions
- Game play
- Keeping score
- Determining the highest score
- Starting a new game

These all work together to create the complete game. Each section can be worked on independently, even by different people, until it is working correctly, and then it will be tested with the rest of the modules already written.

Be sure you understand these steps and follow them when you are working on your Create performance task. You will have a better product as a result!

Documentation

Program documentation is the "guide" to how the program is designed to work. The requirements document created in the investigation phase is the start of the program documentation. Programmers should also add to the program documentation, explaining the program features and how it should be used to meet the functionality originally identified in the specifications. This also could be in the form of a user guide to use in training those who will use the programs as well as "help" text. Ideally, documentation will be created as the program is being developed by the programmers to accurately record what a program is intended to do.

Well-named variables and procedures can be somewhat "self documenting" by describing their purpose, but do not take the place of additional documentation. Documentation is so useful when you either return later to modify the code or a team member has to pick up your code and understand what it is doing to successfully make modifications.

Programmers should comment blocks of code inside the program to indicate the functionality of those sections. Remember that comments are ignored when the program is translated into machine code and are to help people understand what is going on in a program. Comments should also be used in a program to explain complex or confusing sections of code. Note that not all programming IDEs allow comments, so the program documentation becomes even more important in these cases.

Another important topic to include both in your code as a comment and in the project documentation is crediting any code used from another source. This can be a program library imported into the program or through an API (Application Programming Interface) that allows connectivity to an external program. This is important for giving credit to the author as well as stating where the code originated if changes need to be made that will impact the use of the code.

and personal use. Programs can include video, audio, images, text, buttons to push, and items to drag and swipe.

Algorithms

An algorithm is a set of steps to do a task. Recipes are algorithms. They are a set of steps to prepare food. Instructions for taking medicine make up an algorithm. Directions to get from one location to another are an algorithm.

In computer science, algorithms are the set of steps to solve a problem or complete a task. Algorithms are implemented with software. Examples are programs to:

- Calculate grade averages
- Run the air conditioner when the room temperature reaches 78°F
- Calculate the shortest route from home to school on your GPS

Algorithms have the potential to solve many problems once the programs are written for them to run. For example, a sorting algorithm is simple conceptually. Once it is programmed, an indefinite number of datasets can be sorted using it. As improvements are made to software and hardware, additional programming implementations are now feasible. For example, large datasets can now be handled by distributing the data over multiple programs or multiple devices, each processing a section of the code or data at the same time.

A section of code may work independently or can be used with other programming modules. These program modules read in values, make computations as needed, and produce output to automate processes. Programs may have a variety of data coming in (input) and going out (output). Input or output data could be in the form of text, video or images, audio files, or other formats, depending on the needs of the programming solution.

Languages for Algorithms

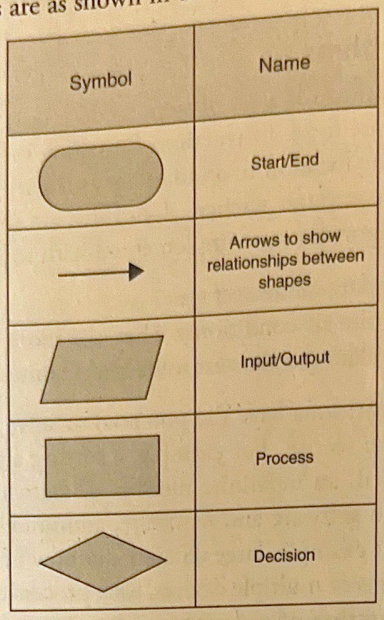
Algorithms can be written in several ways. Natural language is our native speaking and writing language, so it is much easier for people to use and understand.

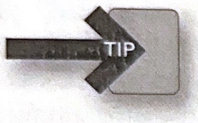
Programming languages are very strict with their syntax, which is like their grammar and structure. This includes both block-style programming languages and textual programming languages. These are written for the computer to execute and may be more difficult for beginning programmers to understand. Pseudocode is used to map out a program's structure before beginning to write the code and uses a combination of natural and programming languages. As you first start out designing algorithms, you will likely use more natural language features. As you learn more about a programming language, your pseudocode may begin to include more coding-like features. Pseudocode cannot run on computers and needs to be translated to a programming language to do so. While most programming languages are fairly equitable in being able to implement an algorithm in the code, some were written for specific uses in a particular subject area, such as physics, and are better suited for these uses.

Flowcharts

Diagramming an algorithm before programming it is another way, like pseudocode, to plan out your program prior to starting to code it. Flowcharting helps to visualize how the program will be structured. You may see places for loops and procedures at this early stage. In some ways, it's like doing a puzzle. You can see how requirements are met with the flowchart, and how some may work better in a certain order or combination in your future code.

There are predefined shapes used by programmers for creating flowcharts. While there are many shapes, the basics are as shown in the following chart.





Many software programs, including word processing software, include flowcharting shapes. There are other programs written specifically for creating flowcharts. While you won't have to create a flowchart on the AP exam, you might have to view one and answer questions about the content of it.

Using Algorithms

Algorithm Readability and Clarity

An important feature of any algorithm and program is how easy it is to read and follow. Hand in hand with readability is clarity. Clarity refers to how easy it is to understand. Generally, a program that is readable is clear. Readability is important to help programmers understand a program. The original programmer may not be the person who makes changes to it later. Even if it is the original author, if a period of time exists between writing it and modifying it, readability and clarity will impact the time and ability of the programmer to revisit and remember the details about the program before modifying or correcting the code. Features of readability and clarity include variables and procedures that are named according to their use and effective documentation of the program with comments in the code along with blank lines to separate sections of code.

Always be aware that there is more than one correct algorithm to solve a problem at hand or to create something new. Finding different algorithmic solutions can be helpful in identifying new insights about the problem. Different algorithms may also have different levels of efficiency or clarity.

Foundations of Computer Programming

Variables

Variables are placeholders for values a program needs to use. A program may need variables for numbers, both integers and real numbers, as well as text fields and Boolean values. Programs can assign values to variables as well as update the value to be a new one.