

## PART 1 INTRODUCTION TO PROGRAMMING

### CHAPTER 1 UNDERSTANDING PROGRAM LOGIC AND PROGRAMMING

#### IN THIS CHAPTER, YOU WILL LEARN HOW TO:

1. Define the meaning program logic and a computer program.
2. Describe the relationship between computer programming logic and computer programs.
3. Explain the process of solving a problem with program logic and a computer program.
4. Describe why program logic is modeled with pseudo code and flow charts before being transformed into a program.
5. Recognize the importance of program development methodologies like the program development process and the system development life cycle to develop computer
6. Describe the roles and responsibilities of programmers and the processes programmers use to create programs.
7. Differentiate between compilers, interpreters and virtual run-time machines in regards to building and executing programs.

#### WELCOME TO COMPUTER PROGRAMMING

I would like to officially welcome you to computer programming. I am excited that you have selected this eBook to learn how to develop your own computer programs. I will give you all the information and examples necessary to understand the basics of computer programming. I will use standardized programming techniques and tools which have long been used by programmers to create quality computer applications. I will incorporate common sense explanations with a variety of examples to make sure the theory is backed up with application of that theory.

Programming logic is the central focus in this eBook along with how they are used by programmers to write efficient and effective computer programs. Every program should start with program logic, which is the blueprint, and end with the coding of

programming language statements. The initial chapters of the eBook will be dedicated to logic structures and specifically, structured programming. In later chapters, we will tackle more challenging programming techniques like object oriented and event driven programming.

As a programmer who has been writing programs for more than twenty years, I have always enjoyed the challenge of programming and I hope that my passion for programming will rub off on you also. For me, the next best thing to writing my own programs is helping a new programmer learn how to write their own programs. There is nothing more rewarding than to see the pride of accomplishment in a student's face when they create their first computer program. Let's get started....

**Computer Programming** – Computer programming is an activity performed by a computer programmer. It always starts with a person. It includes a process of creating logic models which is converted into program language instructions that are later executed on a computer. The goal of the program is to solve a problem or perform a task. Like a recipe provides instructions for a cook, a computer program provides instructions for the computer.

**Program Logic** – Program logic is used by programmers to model the programming language instructions carried out by the computer when the program is executed. Think of program logic as instructions you might give someone on how to make something or perform a task they have never done before. Program logic structure instructions might provide a program decision (i.e. if file does not exist, display message) or a loop through a series of repeated instructions. You might also think of your logic model as a blue print where you have mapped out the instructions but have yet to put them in a program as program language statements.

**Programming Language Statements:** Programming language statements are used to implement program logic by sending instructions to the operating system. is logic with a program built on program language statements.

Programming Tip: Logic models are programming language independent which means logic writing concepts in the eBook can be used across all programming languages.

# A COMPUTER PROGRAM

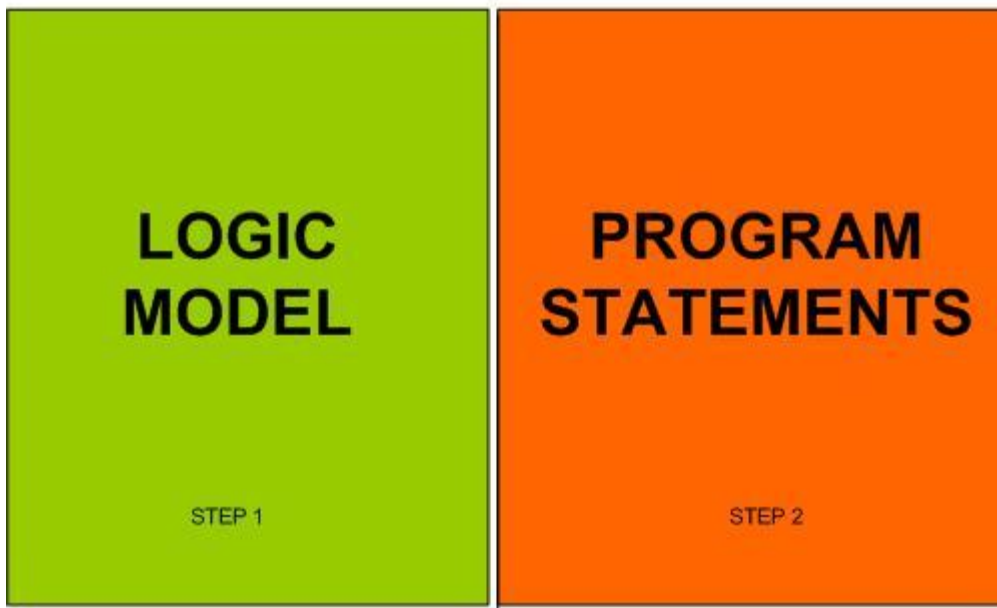


Figure 1: A Computer Program and its Parts

## A DISCIPLINED APPROACH TO PROGRAMMING

When we learn about logic and programming, we will always examine WHY we are using that logic structure or programming methodology. Whenever a new topic is introduced we will stop to discuss WHY that topic best fits the situation we are trying to solve. We will follow this explanation with examples. The examples will include pseudo code, flowcharts, UML (Unified Modeling Language), PYTHON programs, illustrations and case studies. In this wiki, we will use a variety of approaches and different learning styles to provide real life solutions to real business problems. We will apply concepts to case studies and assess our progress to get feedback on in understanding how and why the topics of that chapter are important to becoming successful programmers.

There are many textbooks that teach you how to program. These textbooks cover the syntax and structure of how programs are constructed specific to a particular programming language. One of the more popular programming languages is Microsoft's Visual Basic. In the typical introductory Visual Basic class, the student

learns Visual Basic statements and keywords, how to implement programming logic and structures and finally, how to use the Visual Basic development environment to build and test his program. This approach to learning programming stresses how to create Visual Basic programs but not effective programming logic design. There is a weakness in learning programming with this approach. The emphasis is not on WHY it needs to work a certain way but instead what it will look like. As you'll see in this wiki, the direction will be in formulating efficient, flexible, reusable, high quality programs that follow accepted programming standards.

Since the logic is used to design the solution, you must also understand that construction of logic are key to good programming. Let me illustrate with an example. I think most of us would admit you probably don't have to be an architect to design a house. After all, most of us have had some experience living in a house. Through this experience, we could design what you would find in your typical house (rooms, doors, windows, roof, etc.). I also think most of us would be able to assemble something that looked like a house if given the appropriate building supplies and tools. We could build a house that had a peaked roof, had some windows, had a front door and would have walls and ceilings that would make the house habitable. From a distance, the house might look like every other house on the street. However, would the house be structurally sound? Would the house meet the appropriate construction code regulations? If another plumber or electrician came to this house and had to make a repair, would they be able to? What this house we built meet the professional standards found in the work of professional builders, plumbers and electricians? I don't think so. We know how to build it but not why a floor needs certain supports or why wall studs hold up the ceiling.

This same analogy holds true in first time programming classes. Many times because of time constraints and the amount of material that must be learned, entry level programming classes have to focus on the "HOW" and not the "WHY" of a programming language. "HOW" might get the assignment done and a compiled program but it does not necessarily make you a programmer. Once you know "WHY", you can use the "HOW" to implement the solution. Once we know the "WHY" of programming logic and how to use structured and object oriented fundamentals, we will be able to build programs that follow professional standards for design, and maintainability. Knowing "WHY" we do things is what makes the practitioner a professional. In our case, knowing "WHY" is important in becoming an effective programmer.

## THE PROCESS OF PROGRAMMING

The goal of this eBook is to teach you about programming. Programming is a process and made up of several activities. The program process starts with the modeling of logic. This is also where we will focus much of our instruction. It is often the neglected piece of the programming processes but critical for new programmers. Once the logic is complete programming starts. A completed program must finally be tested before put into production. This eBook was designed to provide instruction on programming for a variety of programming languages (language independent). The process of programming is the same no matter what programming language is used (i.e. Visual Basic.Net, C++, C# or Java).

**Key Concept:** A program is the implementation of your program logic with programming language statements. Program logic and programming come together like a pilot and his airplane. Each by themselves offers little but together, they can provide much more. In short, programming logic is your plan for solving a problem and the program is the implementation of our program logic with programming language statements.

**Program Language Statements** - Program language statements are English like statements that when executed in the correct sequence can instruct the computer to perform a series of tasks. There are many computer programming languages each having differences and similarities. The differences allow some programming languages to work with some computer applications better than others (i.e. the COBOL computer language has long been a favorite of business programmers).

## A FIRST LOOK: UNDERSTANDING PROGRAMMING PROCESS

We must start our understanding of programming by dispelling a few computer myths. Many of us have grown up with a "Hollywood view" of computers. That view has presented the computer as some sort of super brain. A device that we need only ask questions of and it will return to us knowledge we don't possess. A machine that can reason, learn or acquire intelligence on its own.

This could not be any farther from the truth. One of the difficulties in learning how to implement programming logic is the constraints the computer places on us in solving business problems. The human brain is far more complex and the brain's ability to

reason and learn far exceeds that of a computer. Computers are only as smart as the persons programming them and you must condition yourself into providing an instruction (logic) for everything a computer program needs to do. Planning or modeling program logic before we create programming statements is one way for the programmer to condition the solution into one the computer can perform. Without a logic model it is easy for the new programmer to assume the computer can perform more than it is able. We will learn more about logic structures in later chapters but first we will give you a first look.

I often use the example of making a peanut butter sandwich to illustrate the intelligence (or lack of) of the computer. Making a peanut butter sandwich is a task that any three years old could handle. Let's see what we would have to tell the computer in the form of logic instructions to make a peanut butter sandwich. I will accomplish this by documenting the logic necessary with a programming modeling language called pseudo code to build the necessary instructions.

**Pseudo Code:** Pseudo code is a nonstandard English-like programming language used by programmers to model logic and instructions. Pseudo code is called a programming language even though it can not be executed by a computer. It is designed for planning purposes only. By nonstandard, if you open five books on programming, all would have pseudo code examples but all of those examples would look somewhat different. Most computer languages have required standards that must be followed when creating programs. Pseudo code provides us with a way to concentrate on just logic structures without be constrained by the syntax rules of a particular programming language.

**In Practice:** Pseudo code for Creating Programming Logic - Pseudo code has long been used as a mechanism for modeling logic before the logic is implemented as source code statements in a program. We will formally introduce pseudo code later in the wiki. We use pseudo code because it is an excellent way to just concentrate on programming logic without being constrained by the rules of a specific programming language. The beauty of pseudo code is that since it is a list of English instructions, you can easily document the process steps and validate the steps with others. Even persons with no formal understanding of programming logic or programming, can participate in the creation of pseudo code. Pseudo code is not like a formal programming language and if you can read you can follow logic documented in pseudo code.

One of the difficulties in teaching logic with pseudo code has been its nonstandard nature and the fact that since it is not a true programming language, the logic statements can not be executed on a computer. By not being able to execute the

logic, it is not possible to validate the program logic to make sure that it was constructed correctly and arrives at the correct solution.

### Status Check:

Why logic is modeled first rather than just writing the program from scratch?

What does it mean when we say pseudo code is language independent?

What do we mean when we say programming is a process?

## PSEUDO CODE FOR A PEANUT BUTTER SANDWICH

Let's set up the logic instructions with the following scenario. You have at your disposal a kitchen robot. This kitchen robot is capable of taking instructions that will allow it to move around a kitchen, find various food ingredients, and mix and prepare those ingredients. Your pseudo code program needs to give the robot the instructions to create a peanut butter sandwich.

- Task 1: Find a peanut butter in Kitchen Pantry
  - Move across the Kitchen to the Kitchen Pantry
  - Does Kitchen Pantry have a door?
  - If yes then, open door
    - If\_count equals number of shelves are in the pantry
    - While count less than or equal to shelf\_count and peanut\_butter not found
      - Search shelf
        - If peanut\_butter found then peanut\_butter equals true
        - Remove peanut\_butter from selected shelf
        - add one to count
  - If peanut\_butter not found check refrigerator

- Move across the Kitchen to the refrigerator
- Open refrigerator door
  - shelf\_count equals number of shelves are in the refrigerator
  - While count less then or equal to shelf\_count and peanut\_butter not found
    - Search shelf
    - If peanut\_butter found then peanut\_butter equals true
    - Remove peanut\_butter from selected shelf
    - add one to count
  - If peanut\_butter not found then Display “No peanut better today Msg.”
- New Task: Open jar
  - Turn peanut\_butter lid counter-clockwise
    - If peanut\_butter jar equals empty then “No peanut better today Msg.”
  - New task: Find Bread
    - more instructions here

New task: Find butter knife to spread Peanut Butter

more instructions here

I think you can see what is happening here. What would be a very easy task for your typical three year old to accomplish becomes very difficult for the computer because of the amount of logic detail we must supply. It’s not so much that programming logic is complicated but more because of the amount of detail we have to build into our instructions. Our brain is much more sophisticated and powerful and works much differently than a computer. Because of this, we have to break down even simple tasks to a much lower level so that the computer can process them successfully.



**Logic Tip:** Most students will initially have a problem with program logic because they are not familiar to the very unnatural process of breaking down instructions to the computer at that low a level. We will be the chance to learn more about logic in later chapters.

After seeing how complex a peanut butter sandwich becomes when documented as logic steps, was it the computer that caused the problem or was it the programmer of the computer application? What was needed was additional program logic in the payment processing program that would have softened the penalty if the under payment was less \$10.00 dollars. Unfortunately, blaming the computer is very easy but also very incorrect (not to mention the additional cost associated with losing a previously loyal customer). Now I've let you in on that secret. The real brain behind the computer program is the person who programmed it.

**Related Subject:** Blame it on the computer – “The computer did it!”: Have any of you ever experienced this? An organization you do business with has made a mistake concerning your account (i.e. they forgot to credit a returned item to your monthly statement). Their response to the error is, “the computer did it. It is not our fault!”

I can fondly remember one day receiving a rather threatening notice from a local department store because my payment had mistakenly been processed \$2.00 short of the monthly invoice amount. Needless to say, I wasn't happy when they threatened to turn me into a credit agency for two dollars. I called the customer service department to communicate my dissatisfaction. The person on the other end of the phone, although apologetic, could only say that my problem was the result of the computer causing an error. No person there was certainly at fault. “You know these computers, they are always making errors.” The computer was cause of my problems. The computer was the one I should direct my displeasure to. I asked if I could talk to the computer. The phone went silent.

---

## PROGRAM LOGIC AND AN APPLICATION COMPUTER PROGRAM:

In this wiki, you will see repeated references to program logic and application computer programs. We are focusing on program logic in this eBook but it is important that we also refer to programs since the two subjects are so closely tied to one another. We will do a more formal introduction on programming in the next

section but look at the relationship between program logic and a computer program in the following diagram (see Figure 1). Note how programming languages also use English like statements to represent instructions to the computer. Computers do not speak English (although I talk to mine all of the time).

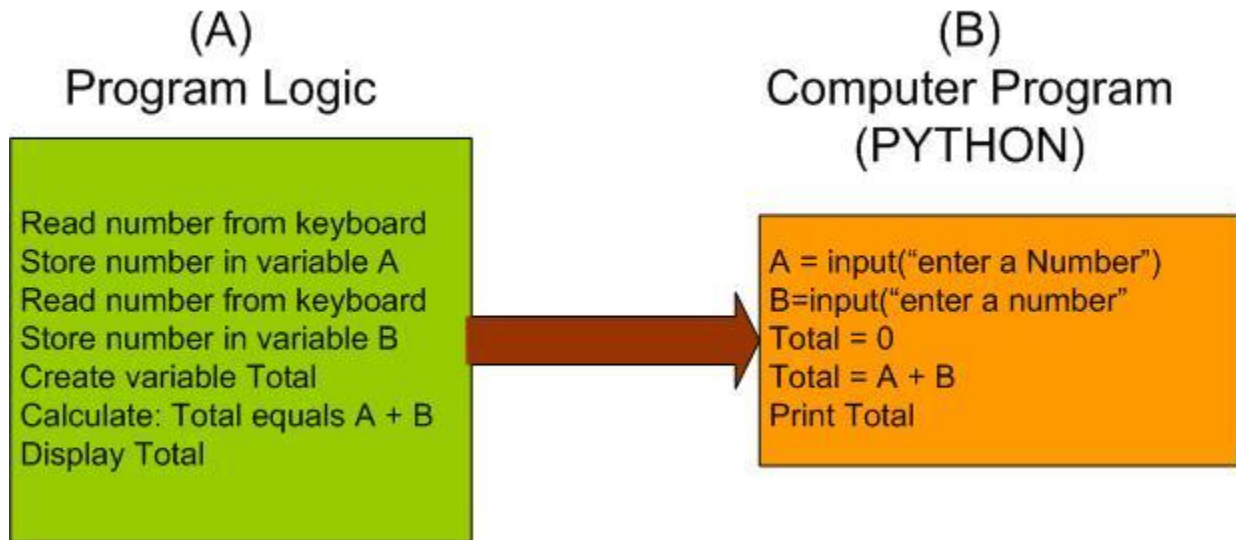


Figure 2: The graphic represents the relationship between program logic and a computer program. Pseudo code statements from Box A are converted by the program to program language statements (PYTHON) shown in Box B

---

#### AN EXAMPLE: PROGRAM LOGIC TO ADD TO NUMBERS TOGETHER

We will use four approaches to model program logic. The simplest approach is pseudo code. As we get deeper into the eBook and begin our learning on structured programming, we will introduce flow charting. Flowcharting is the most graphical of the three approaches we will cover and we will introduce flow charting in a later chapter.

**Flow Chart** – a flow chart is a standardized technique for graphically representing graphics with a set of standardized symbols that represent program logic. To read the sequence of logic steps in this flow chart, just follow the letters.

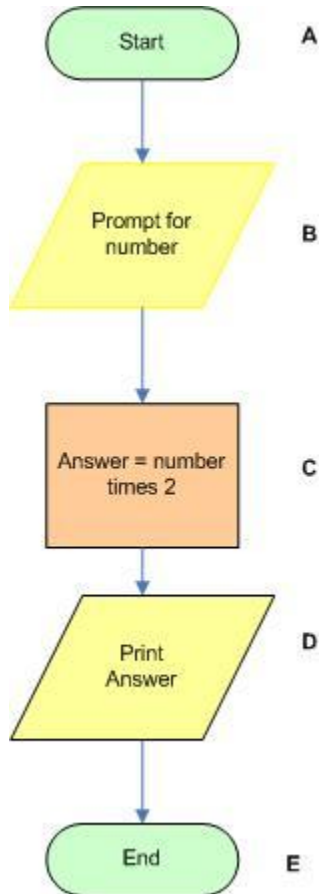


Figure 3: Sample Flow Chart

## A COMPUTER PROGRAM DEFINED

I can still remember the day when I was asked by the Leader of my daughters Girl Scout troop to come in on career day and describe to the other Girl Scouts what I did for a living. Maybe the truest test of understanding a subject matter is to take something that can be as complicated as programming and reduce it to an explanation that even the youngest of students can understand.

For me, my simplest definition of a program is a series of instructions used to solve a problem with a computer. For the Scout troop and for the programming beginner, my explanation is it's like a recipe for chocolate chip cookies. For most of us, we start with a recipe. A chocolate chip cookie recipe provides a set of instructions that tell you what ingredients are needed, how each ingredient should be prepared and the sequence in which each ingredient needs to be combined to make the cookie dough. There are also instructions in the recipe on the temperature of the oven along with the time it takes for the cookie to cook. A recipe has everything you find in a computer program.

### **Status Check**

Why is the computer program only as good as the programmer programming it?

What are some of the advantages of using a programming language like PYTHON to help learn about program logic?

Can you define and differentiate between program logic and programming?

## TYPES OF COMPUTER PROGRAMS

Now would be a good time to introduce you to some different types of computer programs. Yes, not all computer programs are the same. Our example program (adding two numbers together) was a custom application program used for a specific business problem (adding two numbers together). This is not the only type of program. The Windows XP operating system program that came with our PC is also a computer program. It differs in both complexity and purpose but it is still a program. Even though these programs have a very different purpose, they were all built from a logic model and converted into computer programming language statements.

Most computer programs fall under three classifications. These classifications are operating systems, packaged applications (also called shrink wrapped applications because they are sold in the store boxed in plastic) and custom applications which are written to solve a specific problem. The applications we will create and execute in this eBook are custom applications since they are created as business applications for individual use.

Operating system software like Microsoft Windows or Linux are sophisticated programs which control the computers resources (hardware and connected devices) along with launching application programs (maybe like our PYTHON program we had seen earlier) or from an application program like Microsoft Word. The operating system is perhaps the most important applications program on our computer because without it, other programs could not be executed and the attached hardware is in accessible.

**Key Concepts:** Operating systems allow for the execution of applications programs.

Packaged software are application programs which are written to solve a particular type of generic problem. For example, a spreadsheet application is a program which is designed to provide a matrix of rows and columns where data and expressions can be inserted and manipulated. The program has a generic quality that allows it to be used for a number of different activities related to manipulating and modeling numbers. Spreadsheets are very popular with people who work with accounting tasks. Packaged software application programs need to have an operating system installed before they can be executed.

A final type of program is a custom application program. A custom application is a program which has been developed specifically for a particular problem or process. For example, if I was a manufacturing company, I may want a program that specifically manages the inventory of a product which I produce. This product and the process that develops it may be so unique that a packaged software application does not exist. By creating my own software application, I can create a competitive advantage over my competitors by offering my product at a lower cost because of the development of this custom application (it costs me less to manage inventory because of the application that was developed). Custom application programs need to have an operating system installed before they can be executed.

---

## WORKING LIKE A PROGRAMMER: THE PROCESS OF CREATING PROGRAMS

**In Practice:** SDLC – System Development Life Cycle in Detail: SDLC stands for system development life cycle. It is a methodology/process used by people who work in information technology to solve problems to providing effective and efficient

solutions. It remains as the foundation for planning and implementing most systems projects (i.e. application programming). If not for programming, SDLC can also be used for hardware and networking projects. It represents a common tool that you will frequently encounter in working with computer systems.

System Development Life Cycle: I tell students that using SDLC is an exercise in common sense. Do not let the term intimidate you. It sounds very complicated but in fact is a very logical way of approaching and solving a complex problem. In short, SDLC is a "rifle approach" (versus an unorganized shot-gun approach) to quickly and successfully solve a problem. It includes four common sense process steps to solve any type of problem (1) define the problem and assign it a priority (2) analyze the problem particulars and identify possible solutions, (3) design a solution and (4) implement your solution. This process is not just for computer programming. It should work on everything from fixing a flat tire to building an operating system. It works because it documents a plan of attack that logically focuses on a problem and identifies alternatives.

---

#### SDLC CONTAINS THE FOLLOWING PHASES:

1. Planning - Defining the problem and prioritizing against other programming projects
2. Requirement Analysis - Fact Finding problem specifics through observation, questionnaires and interviews.
3. Design - Develop problem alternatives. and models to represent problem and solution
4. Development - Coding and testing of the program
5. Implementation - Installing the program and training the user
6. Maintenance - On-going changes to the program (maintenance)

SDLC contains a series of steps that must be completed as phases. For example, the first phase needs to be started before the second phase and the third phase is to be started before the fourth phase and so on. Even though there is this dependency there is also no reason why there cannot be parallel activity. By this I mean one phase can start before the preceding phase is completely finished. SDLC has been called a "waterfall" approach because one phase flows into the next phase.

**Key Concept:** SDLC can loosely be defined as exercising common sense. Most problems, computer or non-computer, can be solved by using a logical common sense approach to solving the problem. After all, aren't most problems solved by getting the facts, designing a solution and implementing the solution? It could be a math problem, a science problem or a problem with your car, the approached suggested in SDLC could work for them all.

## PROGRAM DEVELOPMENT PROCESS

To become a successful programmer, you need to begin to work like a programmer. To work like a programmer, you have to start using the program development process. Even though the process name sounds rather imposing, the program development process (PDP) is actually very straightforward and very much common sense. PDP is a sub-process of SDLC that becomes part of several of the steps of SDLC.

**Program Development Process (PDP)** - you can think of the program development process as standardized set of steps used to provide a logical, common sense approach (or process) to solving a difficult problem with a computer application. PDP is a checklist that ensures that the steps of building a program are followed so that no important step is omitted or taken out of order. This process has long been used by programmers as a professional standard because of its simplicity, effectiveness and consistency.

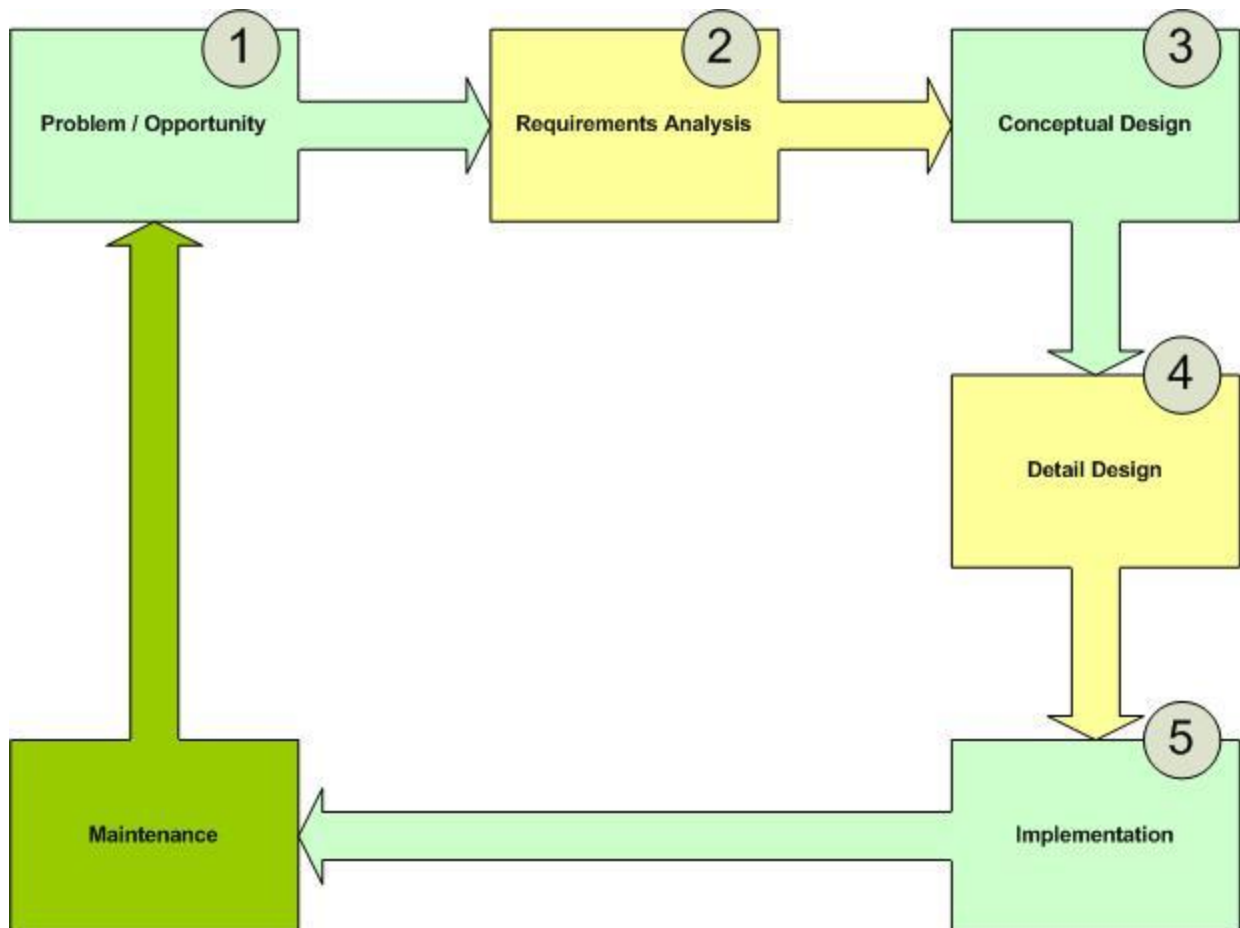


Figure 4: Program Development Process is outlined as a five step sequential approach. Maintenance is reflected as a separate process that can occur one or more times after the original program is implemented.

**Key Point:** SDLC and the PDP are not industry standardized processes. In other words, you may find different various versions of each depending on the development process you are using. The PDP, in particular, is a process created specifically for the eBook as a composite of other similar program development processes used in the industry. If your SDLC and PDP phases look different than in other textbooks, look at the activities in each phase and you will see that even if the phase name is different, the spirit of each phase is the same.

## Status Check:

What are the different types of computer programs?

Can you execute an application program without an operating system?

Why was it written that we are concentrating on custom applications in this wiki?

What are the phases of SDLC?

How is PDP similar to SDLC?

## AN EXAMPLE: USING THE SDLC AND THE PROGRAM DEVELOPMENT PROCESS

PROBLEM: PROGRAM LOGIC IS NEEDED TO DEVELOP A PROGRAM THAT WILL AVERAGE CLASS GRADES.

### Program Development Process Phases

#### PLANNING

Start with a problem statement: I need a program that will calculate the test scores for a class of 15 students and tell me the average test score.

#### REQUIREMENTS ANALYSIS

\*Fact finding or requirements analysis (done by interviewing the end user): Do you have an example of how you want the average number displayed? Do all of the test scores need to be displayed or just the final average? How would you like to input the scores? One at a time or all at once? What if there are less than 15 scores?

#### Design

- Conceptual design: Conceptual design usually takes the form of a logic model that lays out the ideal logic structures to provide a solution. In conceptual design, we will program the expressions needed to calculate the average along with the logic necessary to accept the inputted values.

- Detail design: We are ready to code our solution using a programming language. We take the conceptual design as a starting point and using the syntax options of the programming language, code the logic structures into programming language structures.
- Testing: We will perform unit tests as we go. Unit testing is used to test sections of the entire program. A system test will be performed at the end of the process to make sure our scores are calculating correctly and that any errors are flagged for the end user.

---

## IMPLEMENTATION

- Implementation: During implementation, we move into production the test average calculator program on to the PC of the person using it. The program is complete and ready to use.

---

## MAINTENANCE

- Maintenance: No program is written once and not revisited without some changes. No sooner than the user started to use this program then did they decide that it would be useful in their other classes. The other classes have different numbers of students so that a program that score 15 tests needs to be changed to a program that scores however many tests the user has to grade. We now start back up at the problem statement and move through the phases once more for this program maintenance.

## OTHER PROGRAM DEVELOPMENT PROCESS CONCERNS

The following processes and activities are not defined specifically to one program development phase but in fact can be a part of several or all of the PDP phases.

---

### DEPLOYMENT:

Deployment concerns the techniques that will be used to deliver your program to your users. Deployment is affected by options available for the distribution of software (i.e. some user's computers will have to be updated by CD-ROM or updated via the Intranet/Internet) and the type of program (Internet or Desktop). You'll also want to consider if automatic configuration is required by your program. This might include placement of an icon on the users desktop or placement of the program in the programs start menu (in the case of Microsoft Windows).

Internet applications have different deployment concerns and require planning throughout the PDP process. Programs do exist for the programmer to assist in deployment. Understanding the features and functions of these products early in the detail design process can make deployment much easier when the project reaches implementation.

---

#### TRAINING/DOCUMENTATION:

It is very easy to forget about training and documentation until the project is complete. Many times training and documentation is either ignored or given no consideration at all. You need to look at this task as one of the ways to ensure your programs succeed. If users cannot understand how the program works, they will both reject the program out of frustration or turnaround and request a number of modifications which will only lengthen the amount of time and cost needed to complete the software.

**Programming Tip:** Many programming languages support special documentation tags (these tags are similar to HTML tags, if you are familiar with web page formatting) which can be added during the coding process so that documentation can be created automatically when the program is complete.

Training is sometimes the responsibility of the programmer and is sometimes the responsibility of education and training staff. Even if it's the responsibility of another group, as the programmer, you'll certainly have to be actively involved in describing how the program operates and what prerequisite skills will be necessary for full use of the program's features.

---

#### PROJECT MANAGEMENT:

Not all programs are contained in a single file and programmed by one programmer. Depending on the size of the application project, you may be part of a team who is developing a new system. This team may be developing 20 or 30 programs independently of one another but programs which will need to function in concert with one another at the project's conclusion. To facilitate an effort of this magnitude,

typically a project manager is assigned to the project. The project manager will likely use project management software. Project management software packages are designed to improve the productivity of the entire team by keeping track of who's doing what and in what sequence. It helps ensure that programs are developed on schedule and that one programmer is not held up waiting for the work of another programmer to be completed.

**Status Check:**

What are the main tasks in conceptual design phase of PDP?

Why is deployment, project management and training outside the tasks in PDP?

## CHAPTER CASE STUDY: CACTUS BOOKS AND TAPES

Each chapter will have a common case study to integrate topics covered in the chapter. Cactus Books and Tapes is a fictitious small business run by two sisters who are just starting to incorporate technology to meet their growth. Like most small businesses they are trying to use Internet to improve customer service and increase sales. The case study will also parallel the assignment cases studies found at the end of each chapter.

---

### CACTUS BOOKS AND TAPES

Cactus Books and Tapes (CBT) was opened in 1998 by two sisters who saw a business opportunity in the sales and rental of audio tapes. The business began with a small store in a strip mall near a large office complex. The traffic to the store has always been good and because of the prudent planning of Christina and Lois Chavez, the store expanded from the selling of tapes to also books. The theme of the bookstore is still focused on self-help materials but the Chavez sisters carry best sellers and do a good business in custom orders of books and tapes. The store added a new location last year and the business continues to grow. The new store includes a coffee bar and an area for customers to read and enjoy a cup of their favorite tea or coffee. Many of the CBT customers would like to request custom book orders on the Internet instead of driving to the store or having to worry about someone being available to answer the phone. The sisters see the development of a web page to accomplish this as a critical project for continued store success.

The store is organized as a partnership between the two sisters. The business has combined retail space of 25,000 square ft (10,000 feet at first location and 15,000 square feet at the new location). Christina manages the original location and supervises the accounting for the business (accounts payable, accounts receivable, payroll and human resources). Lois manages the new store and has been focusing her attention on managing the stores new coffee bar. The business has two assistant managers at each store (day and evening shifts) and has approximately 35 employees combined. The store is open 6 days a week from 9 a.m. to 9 p.m... Sales volume from both stores is approximately \$750,000 dollars per year.

Since neither Christina nor Lois has extensive computer experience they have contracted your expertise as an outsourced employee who will help them in the design of their Internet book order service. Outsourcing is an excellent way for an employer to be able to work with and evaluate the knowledge of a potential new hire. Since CBT has a good benefits plan and competitive salaries, this is an excellent opportunity for you to get on to the ground floor of a growing small business.

Your first challenge is to map out a plan of action so that the design of the Internet book order service will be successful. You will use the steps outlined in the program development process and with SDLC to let the Chavez sisters know what steps need to be performed to complete the project.

To do this you have decided to use an SDLC diagram to provide a high level picture of the overall plan and the program development process to outline the specifics required to build the programs used to capture Internet book orders.

## USING A PROGRAMMING LANGUAGE

Taking your program logic and converting it into a compiled program so that it can be executed on the PC requires several steps and various tools. First and foremost, the programmer must decide which computer language they will implement their solution in. Each computer language is similar and is different. Most computer languages are similar in that they use common structures (the structures we will be learning in this wiki) and different in that they use different keyword statements and syntax.

**Tip:** The Internet is a great source for researching the different programming languages. Many languages are in the public domain and programming tools exist to learn these languages at no cost (providing you have a personal computer).

#### Background Information: Programming Languages

There are perhaps hundreds of programming languages available to the programmer across various hardware platforms and operating systems. Some of these programming languages were designed for very specific kinds of applications (i.e. scientific or business) and many others or designed to be used in a more general sense (all types of software applications).

Many computer languages have been standardized so that their use can be applied across multiple hardware platforms and operating systems. Popular languages such as Java are available on everything from a supercomputer to a cell phone. The recent trend in new programming languages is to create languages that support structured and object oriented programming (both which will be covered extensively in this wiki) across multiple operating system platforms. For example, in object orientated programming there is a great advantage in being able to create class objects (blocks of logic statements) that can be reused over and over again in multiple programs. Since programming has always been a primarily labor intensive activity, any developments, like code reuse, which allow programmers to create code faster and of higher quality is seen as important. Java and C# (C sharp) are two contemporary languages which were designed specifically for programmer productivity (with code reuse support) and for support across multiple computing environments. Older languages, such as BASIC, have continued to be modified in upgraded to support several of these same code reuse capabilities to also improve programmer productivity.

Most modern programming languages also support program development software to make the programmer more productive. This programmer's software development environment is typically called an IDE. IDE stands for Integrated Development Environment and includes the ability to create and edit source code from program logic, compile code to remove syntax errors in language statements, support a visual debugger which helps to isolate logic errors and finally a linking environment to join compiled modules into an executable program. This sounds complicated but in reality the IDE allows us to author the program and then create a compiled file that has converted our programming statements to codes that the operating system can execute.

The IDE is the programmers' toolbox and contains software that will make the programmer work quicker and more effectively. Although the only computer

language we will specifically use in this eBook is PYTHON, it is appropriate to talk about some of the various ways contemporary computer languages facilitate the creation of executable programs. We will have samples of Java and Visual Basic also included from time to time within the wiki.

**Integrated Development Environment:** Since programming is a labor intensive activity, the programmer needs tools just like a carpenter needs a saw. The IDE provides the tools you need to become a more efficient programmer. The IDE will at a minimum include a programming source code editor, a run-time debugger to help find errors, a compiler to create an executable version of the source code and usually on-line help with programming language documentation.

---

## UNDERSTANDING PROGRAM LANGUAGE STATEMENTS AND SYNTAX

As mentioned earlier, coding your program with program language statements follows your logic model. Once you have the blueprint complete and the logic organized, you are ready to start building with the programming language. The program language will sequence the instructions you are sending to the operating system to perform the tasks of your program.

Program language syntax rules are the rules of grammar for a particular programming language. Some programming languages require that each statement end with a semi colon. Some programming languages have specific requirements as to how variables must be declared and referenced. All programming languages have slightly different syntax rules just as French is different than Spanish is different than English. Just like it takes time to understand the syntax of Spanish, programming language syntax also takes time to learn.

**Background Information:** Troubleshoot Skills and Debugging your Program Logic - Most programmers spend the bulk of their time maintaining existing software. Because of this it is important that you are able to read and understand other programmer's logic and programming statements. It is just as important that you document your programs and use accepted standards to ease the ability of other programmers to understand your work.

Debugging the program is the process of examining program logic for errors. Programming errors and errors in general have been called bugs. The origination of the term goes back to the very first computers when a gypsy moth flew too close to a vacuum tube and caused the computer to fail. Since the problem was truly a bug (insect), the computer operator coined the expression "bug in system" to describe the problem. The word caught on and computer problems have been referred to as bugs ever since then. The act of removing bugs (or problems) from program logic is called debugging. Software that helps to remove problems is called a debugger. A more traditional word for this activity is troubleshooting which means the same as debugging.

As you take your first programming class, experience with programming syntax can be one of the more frustrating activities. It's not unusual to stare at a program statement for fifteen minutes trying to understand the syntax error, only to have a more experienced person find what is missing, in less than 30 seconds. Whereas the English language allows us to have lapses in our use of grammar, a computer program is very specific and as mentioned in an earlier section of this chapter, computers must be told exactly what to do or they will not be able to do it. The compiler and interpreter will report syntax errors to the programmer and will not let program execution continue until the syntax error is fixed. We will see evidence of syntax rules as we work with PYTHON to develop our program logic.

**Programming language Syntax:** All new programmers will remember their first syntax errors with great fondness...NOT! Learning the syntax (or grammar) of a programming language can be compared with learning any new language. It is much easier reading a foreign language than it is to speak it. This is because to speak it we have to have a strong knowledge of the language's syntax. It takes time and it is frustrating but rest assured, we have all been there.

**Related Subject:** Programming Patterns

There has been a lot of work done recently in studying programming patterns. Patterns are blocks of logic (later converted to programming statements) that represent common logic blocks. The logic statements together would represent a task that you would use again. Maybe a pattern might be reading a file until you have reached the last record. Common logic patterns like reading a data file is the type of logic block that would be easily reused across a variety of different programs.

---

EXECUTING THE PROGRAM

Creating a file containing a series of programming language statements is not enough. The operation system can not recognize program statements in their original "English" like format. A source code program must be converted to a format that the operating system can execute before it can be used.

Executing a program is done in two ways. Usually depending on the programming language, your program source code (program language statements) will be either compiled into a new file containing instructions recognizable by the operating system or read line by line by a program interpreter. Rather than creating a new file of compiled program language statements, an interpreter creates no new file but executes each line one at a time.

## INTERPRETER BASICS

Aside from compilers which create executable machine code, many languages also support interpreters. It is important to note, some programming languages can only be executed with an interpreter. What this means is that rather than creating (compiling) a file of machine code instructions executed by the operating system, the interpreter is a program that reads the source code line by line and converts it into machine code instructions. Interpreters offer some advantages when writing new code but have typically been shunned for production environments since interpreted code typically runs slower than compiled machine code. To illustrate this, I would use the analogy where if you are visiting a foreign country and you do not know the language you would have to have every statement you spoke interpreted. You would have to wait for your communication to be interpreted and you would have to wait for the response to be interpreted. With a compiled program, the machine code is directly fed into the operating system since the translation has already completed.

**Executable Machine Code:** Computers do not understand English but they do understand ones and zeros (electrical current that is either on or off). The source code instructions are converted by compilers or interpreters to binary code instructions that the computer can understand. Until the program is converted to machine code, your program can not tell the computer how to do anything.

**Interpreters:** An interpreter translates the programming language source code to binary machine code. A computer interpreter converts a computer language to

machine codes each and every time it executes. It also has the reputation of running programs slower because of the every time translation but has been used by programming environments like Java and Microsoft .NET (virtual runtime interpreters) to allow programs to be run on multiple operating systems with modification ("write once run everywhere".)

---

## COMPILER BASICS

We have already spent some time discussing the compilers importance to the IDE and comparing the compiler to a program interpreter. Compiled code is typically saved into a file with a file extension of .exe. For some programming languages, the .exe file is standalone and requires no other programs or libraries for program execution. For other programming languages, the .exe file is designed to work with other libraries of machine code that must exist on the computer. Most comprehensive IDE's include the packaging and deployment tools that will bundle all the necessary files for the application to run on a different personal computer. This packaging tool would not only include the IDE created .exe file that also any libraries of machine code necessary to run that program.

**Compiler:** Much like the Interpreter converts computer language statements into executable machine code, a compiler also converts source code into machine code. A compiler converts the source code into an .EXE file that can be run by the operating system. Compiled programs run faster than interpreted programs since the source code to machine code conversion was done in advance and place in a file that can be executed by the operating system.

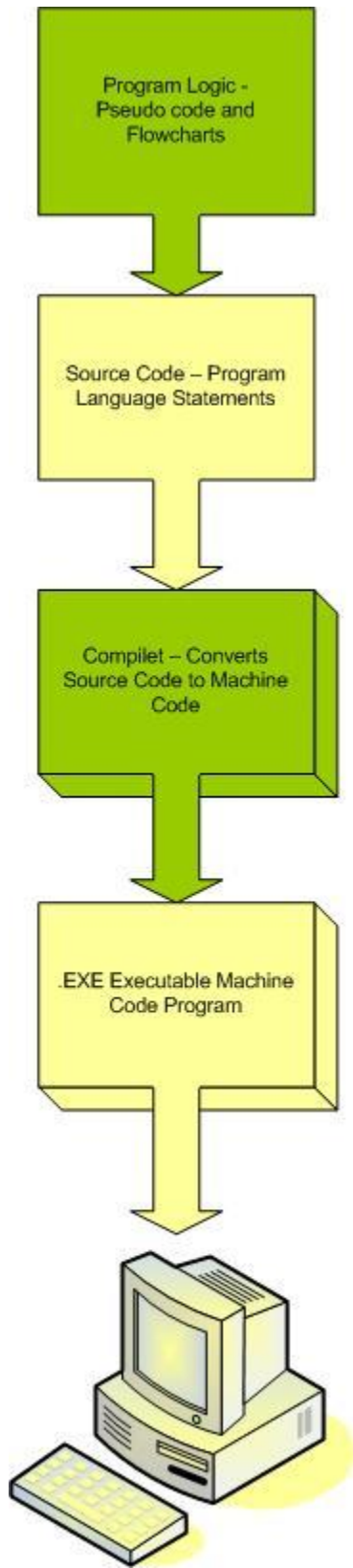


Figure 5: Process of Creating an Executable Program with a Compiler

---

## VIRTUAL RUN-TIME MACHINE FUNDAMENTALS

A virtual runtime machine is somewhere between an interpreter and a compiler. Typically, a program must first be compiled into some intermediate language that will be recognized by an interpreter (also known as a runtime machine). The intermediate language is not quite machine code but has been created in a way to optimize its performance on the interpreter. The Java, Java Virtual Machine (JVM) and Microsoft's Visual Studio .Net Common Language Runtime (CLR) use runtime virtual machines to execute programs.

**Virtual run-time machine:** Here we have a different type of Interpreter. This Interpreter uses pre-compiled files (not compiled for an operating system like a traditional compiler) to send machine code instructions to the operating system. The pre-compiled files for virtual run-time machines can be run on any virtual runtime machine making them runnable on different operating systems.

There are benefits in creating programs that can be executed under a virtual run-time machine. It is now possible to compile a program in a Microsoft Windows environment and run that same compiled file on an Apple Macintosh running the OS 10 operating system. Normally with compiled programs, it would not be possible for the same program to run across multiple operating systems. The term "write once run everywhere" has been used to categorize programming languages that use virtual run-time machines. Both the Java Virtual Machine and the Common Language Runtime have a specification that allows virtual run-time machines to be created for not only computers but also devices like cell phones, personal digital assistants (PDA) and even smartcards. This allows the programmer to use their personal computer to create programs for other computing devices like cell phones and PDA's. This is another way of extending the productivity of the programmer.

### Status Check:

[What benefits and weaknesses of interpreted programs?](#)

How can a programming language that uses a virtual run-time interpreter be an advantage for the software developer?

## PUTTING IT TO USE

The intent of this section is to introduce the new programmer to standards and techniques frequently used by professional programmers. Topics in this section relate to concepts introduced in the chapter but with a more vocational or occupational focus for students considering a career in programming.

## LIFE AS A PROFESSIONAL PROGRAMMER

On a daily basis, programmers are designing new programs or maintaining existing applications. Based on past history, most of the time you'll be maintaining existing programs. Although you'll find in your programming classes that you'll typically be developing a new program with each assignment, most professional programmers are kept busy maintaining existing software. Part of this is because of the labor intensive nature of programming and the cost it takes to develop new systems. Put yourself in the place of the owner of a company or its shareholders, you would want to take a new program and spread its expense over as much time as possible. Just like any investment, a computer program is a company asset and the goal of management should be to maximize the return from that asset. This may mean using an existing program even when a better one could be written if the return on the original program is still favorable. This is the same reason most of us do not buy a new car every year. You will on occasion be involved with the development of new systems. Companies need new software to remain competitive and even existing high quality software sometimes has to be replaced because of new requirements.

**Key Concepts:** Programming is a mix of using your creativity and your problem solving skills to create something from nothing that can be admired just like we admire good music or art.

So from one standpoint it's important that you are able to design new program logic and write effective programs. As a new programmer, it's probably more important that you are able to look at someone else's program logic and understand how to apply the necessary changes.

**Background Information:** A Maintenance Programmer - Your Goal!: I have often told the programming students in my entry level programming classes that my goal (and also theirs) is to have them come out of the class as a competent maintenance programmer. I say this because most of the time this will be what they are going on their first job.

Programming maintenance could involve the repair of an error in the program or the addition of a new program feature.

All professions have activities which are considered benchmarks for success. If you're a major league baseball player than your benchmark for success is probably your batting average or the number of home runs hit. As a programmer, your benchmarks are writing high quality programs that run with the highest standards of performance. Your documentation and design should be clear and concise. Your knowledge of technology is up to date and you're committed to learning new technology as it is developed. Peer respect on the job will come from being able to take problems and systematically and professionally convert them into software solutions.

Programming exercises your mind and challenges you to think. Meeting that challenge and being successful in developing solutions with a computer can be very rewarding. If you ask any programmer how they felt about their career decision, I think most would tell you that they have found it to be an extremely rewarding career. Programmers are considered professional staff. As a professional, their given a great deal of latitude to set their schedule, pick their projects and work with minimal management supervision. Most programmers would tell you the programming is a very good career choice.

**Key Concepts:** Becoming a successful programmer means that one needs to learn the standards and best practices used by successful programmers. These practices include learning techniques for creating program logic. Many times these best practices are not given the appropriate attention in programming classes since programming classes need to focus on the syntax of the language being taught. The focus in this eBook will be on structured an object oriented programming but the examples and exercises will allow you to perform some programming with an easy to use the programming language called Python. Don't be alarmed that you are being exposed to a programming language so soon. Python is a tool that will help you get the kind of feedback necessary to determine if your program logic designs are

correct. It is an added bonus that we will be able to have exposure to a powerful programming language as part of the wiki's coverage.

**Best Practices:** The importance of Good Program Documentation

As I mentioned earlier, most programmers spend a great deal of time maintaining other programs. It doesn't take long for a programmer to appreciate good program documentation. Without program documentation, maintenance on someone else's program could take hours instead of minutes. If the program has unusual logic without program documentation, it seems like it would be easier to build the program from scratch rather than analyze the confusing work of another programmer. All programming languages have a special statement or syntax to leave comments in a program. These comments might be a paragraph to explain a particularly tricky piece of logic or just a statement to accent the timing of the logic.

**TIP:** Even though it is easy to ignore documentation when one is deep into coding a program, it is much easier to place program comments in your code as you go then it is to go back later and try to remember what was important about that part of the program logic. There is probably no such thing as too much program documentation unless the programming language is interpreted in which case the program comments should be placed the end of the program after the programming language statements.

I have included below some sample program documentation for a PYTHON program. The # signs are used in PYTHON to add program comments to a source code. The # sign (along with the text to the right of the #) is ignored by the interpreter and not executed. There are typically no rules or standards regarding comments but most programmers will usually add a sentence or two for a quick explanation.

```
#Gary Marrer
```

```
#Chapter 1 - Comment Example
```

```
#variable address stores a customer's address
```

```
address = "1 Main Street"
```

```
#Subtract cost from sales to calculate profit
```

```
profit = cost - sales
```

It is important to note that I am talking of programming documentation in this section of the chapter. There are other forms of documentation that you may be involved with as the programmer. You may be involved with user documentation for the end user or operator documentation for the Information Services staff. Many organizations have documentation standards which all staff must abide by.

**Related Subject:** The Myth of Self Documenting Code - Many programmers who ignore good program documentation will use the excuse that the programming language is one that is self-documenting. This is especially true of programming languages that have extensive libraries of built in routines that cover most of the common programming logic scenarios. This is a misconception. Even languages with a limited set of statements and keywords need to have program documentation added (via comments) to ensure that the logic of the program is understood when the program is revisited at a later date.

## CHAPTER REVIEW

Chapter summary, highlights, key terms, short answer questions, quizzes and case studies to reinforce topics covered in this chapter.

## CHAPTER SUMMARY

After finishing this chapter you should understand and in some cases, demonstrate the following topics:

- Define the meaning program logic and a computer program.
  - Computer logic is the conceptual design of a program solution. Once the conceptual design is complete the logic is converted into a source code program developed with a programming language to implement the solution on the computer.
- Describe the relationship between computer programming logic and computer programs.
  - Problems solved on the computer with computer programs first start with a plan that is documented as logic in logic models.

- Explain the process of solving a problem with program logic and a computer program.
  - The process of programming requires that you (1) identify the problem, (2) model the solution, (3) implement the solution with a programming language and (4) run the program to derive the solution.
- Describe why program logic is modeled with pseudo code and flow charts before being transformed into a program.
  - In absence of a plan to design a solution, a design leave out important details to built the programming instructions necessary to write the program.
- Recognize the importance of program development methodologies like the program development process and the system development life cycle to develop computer programs.
  - SDLC has long been used to guide systems staff through a systematic process for problem management. With SDLC the process phases are defined as planning, requirements analysis, design, development, implementation and maintenance.
  - PDP is a program development process specifically created for program development. Like SDLC, PDP is a process with a set of phases. The phases of PDP are problem statement, requirements analysis, conceptual design, detailed design and implementation
- Describe the roles and responsibilities of programmers and the processes programmers use to create programs.
  - Programmers write programs and are an important part of very organization and recognized as a professional with limited need for direct management
  - Programmers use tools like Integrated Development Environments (IDE) to make them more productive in what has traditionally been a labor intensive activity.
- Differentiate between compilers, interpreters and virtual run-time machines in regards to building and executing programs.
  - Programs need to be compiled to remove computer programming language syntax errors and to create a machine executable file of the programs instructions.
  - Some programming languages use interpreters or virtual run-time machines to convert programming language source code statements into machine executable instructions executed by the operating system.

## CHAPTER KEY TERMS

Compiler

Computer Programs (Program)

Computer Programming (Programs)

Debugger

Executable Machine Code (.EXE)

Flow Charting

Integrated Development Environment

Interpreters

Program Documentation

Program Language Syntax

Programmer

Programming Languages

Programming Logic Structures

Pseudo Code

Runtime Virtual Machine