Revised: 7/27/2020

Introduction to Computer Programming

Michael C. Hackett

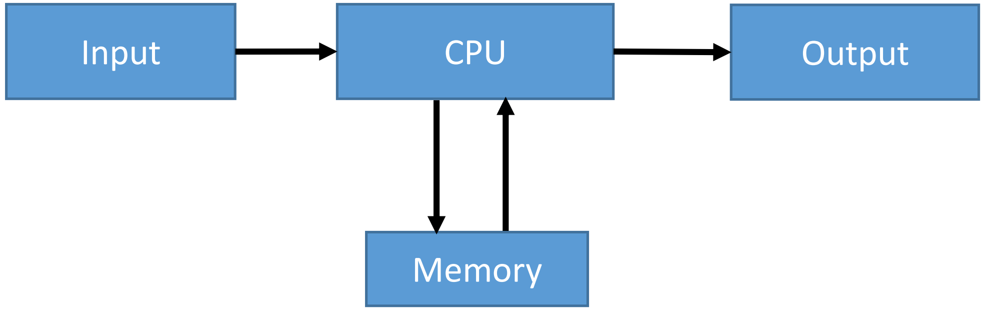
Assistant Professor, Computer Science

Community College of Philadelphia

What is a computer?

- A ***computer*** is a device or machine that is capable of performing arithmetic and/or logical operations.
  - A modern definition would include the capability of storing and processing information.

- A modern computer system is comprised of:
  - Central Processing Unit
  - Memory
  - Input
  - Output



Hardware and Software

- ***Hardware*** is any component of a computer that you can physically touch.
  - Processors, disk drives, RAM, monitors, keyboards, and mice.

- ***Software*** is any intangible component of a computer.
  - Operating systems, applications, pictures, videos, and files.

Major Hardware Components of a Computer

- The Central Processing Unit (CPU)
- Memory
- Input Devices
- Output Devices

Central Processing Unit (CPU)

- The Central Processing Unit is a piece of computer hardware that performs the instructions of computer programs.

- Modern CPUs are microprocessors- a component with a CPU on a single integrated circuit.

- Performs logical and arithmetic operations.

Main Memory

- The system's Main Memory stores program instructions and data that are currently in use.
  - Typically refers to a computer's random access memory (RAM)

- RAM is volatile memory.
  - Data stored in RAM is lost when the chip is no longer powered.

Secondary Storage

- Secondary Storage refers to devices that can store data (almost) indefinitely.

- Secondary Storage Devices, like hard drives, are a form of non-volatile memory.
  - The data is retained even when the device is powered down.

Input Devices

- An Input Device is a piece of equipment that allows information or data to be given to the system by a user.

- Keyboards, mice, webcams, and microphones are all examples of input devices.

Output Devices

- An Output Device is a piece of equipment that allows information or data to be retrieved from the system and presented to the user.

- Monitors, speakers and printers are all examples of output devices.

Other Important Hardware Components

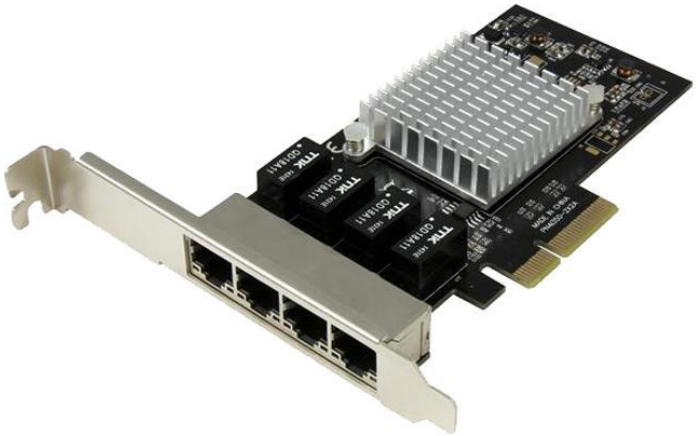
- Motherboard
  - A printed circuit board (PCB) that all hardware components connect to.
  - Allows the hardware components to send and receive data to and from each other.

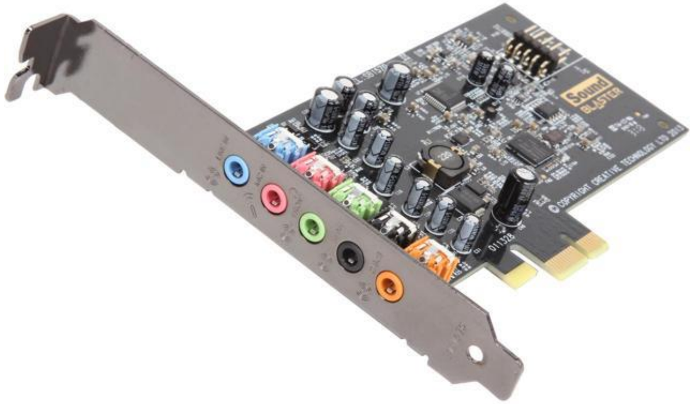
Other Important Hardware Components

- Power Supply Unit
  - Sometimes abbreviated as "PSU" or simply referred to as the Power Supply.
  - Provides electrical power to hardware components.

Other Important Hardware Components

- Expansion Cards
  - Printed circuit boards that are inserted directly to the motherboard.
  - Examples include graphics cards, sound cards and network interface cards.





Other Important Hardware Components

- Cooling Systems
  - Computers typically use fans to circulate air and keep the hardware components cool.
  - Some high performance computers have liquid cooling systems.

Major Types of Software

- Application Software
  - Programs that make your computer useful.
  - Word processors, Internet browsers, video games and mobile apps

- System Software
  - Programs that control the computer.
  - Operating Systems, device drivers, utility programs and software development tools.

What is a computer programming language?

- A ***programming language*** is a formal language that consists of a set of instructions that cause a computer to execute a series of operations or tasks.
  - A group of instructions that completes some task is a ***computer program***.

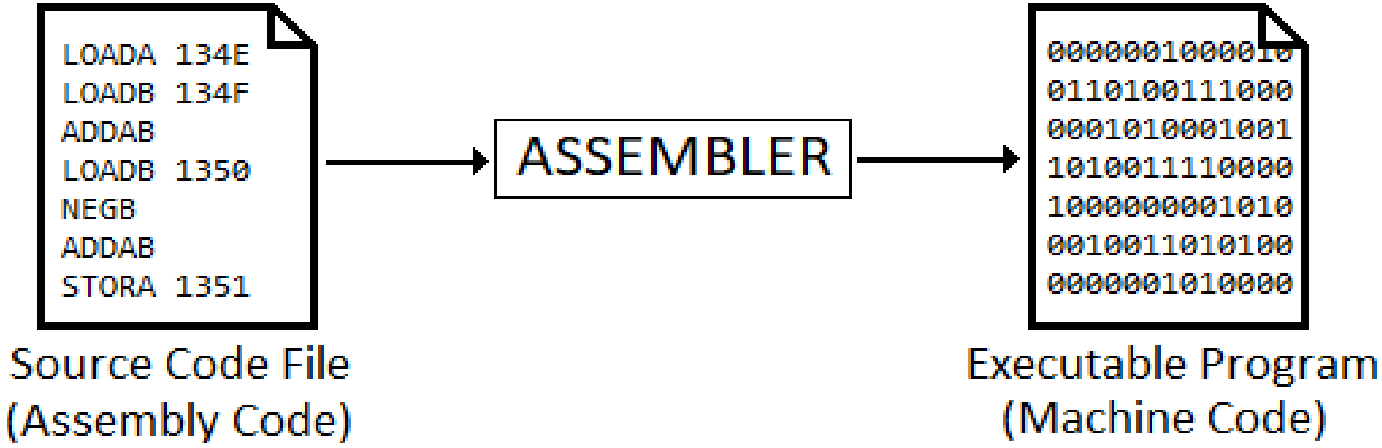
- There are, in general, two major types of programming languages: low-level and high-level.

What is a low-level programming language?

- A ***low-level programming language*** is one where the instructions are (or closely related to) the instructions for the processor/CPU.
  - The language may only work for a specific processor or other hardware.
- Usually refers to assembly language or machine code.
- Difficult to program with.
  - With machine code, it's typically done in binary.
  - Assembly language maps the binary instructions to somewhat less vague instructions. An *assembler* translates those instructions back to binary.

```
; Global declarations
STATUS      equu   3       ; Status register is File 3
C           equ    0       ; Carry/Not Borrow flag is bit0
            cblock 20h
             NUM:2         ; Number: high byte, low byte
            endc
MAIN        goto   SQR_ROOT
; ***************************************************************
; * FUNCTION: Calculates the square root of a 16-bit integer    *
; * EXAMPLE : Number = FFFFh (65,535d), Root = FFh (255d)       *
; * ENTRY   : Number in File NUM:NUM+1                          *
; * EXIT    : Root in W. NUM:NUM+1; I:I+1 and COUNT altered     *
; ***************************************************************
; Local declarations
         cblock
          I:2, COUNT       ; Magic number hi:lo byte & loop count
         endc
         org    200h       ; Code to begin @ 200h in Program store
SQR_ROOT clrf   COUNT      ; Task 1: Zero loop count
         clrf   I          ; Task 2: Set magic number I to one
         clrf   I+1
         incf   I+1,f
; Task 3: DO
SQR_LOOP movf   I+1,w      ; Task 3(a): Number - I
         subwf  NUM+1,f    ; Subtract lo byte I from lo byte Num
         movf   I,w        ; Get high byte magic number
         btfss  STATUS,C   ; Skip if No Borrow out
          addlw 1          ; Return borrow
         subwf  NUM,f      ; Subtract high bytes
; Task 3(b): IF underflow THEN exit
         btfss  STATUS,C   ; IF No Borrow THEN continue
          goto  SQR_END    ; ELSE the process is complete
         incf   COUNT,f    ; Task 3(c): ELSE inc loop count
         movf   I+1,w      ; Task 3(d): Add 2 to the magic number
         addlw  2
         btfsc  STATUS,C   ; IF no carry THEN done
          incf  I,f        ; ELSE add carry to upper byte I
         movwf  I+1
         goto   SQR_LOOP
SQR_END  movf   COUNT,w    ; Task 4: Return loop count as the root
         return
         end
```

Assembly Language

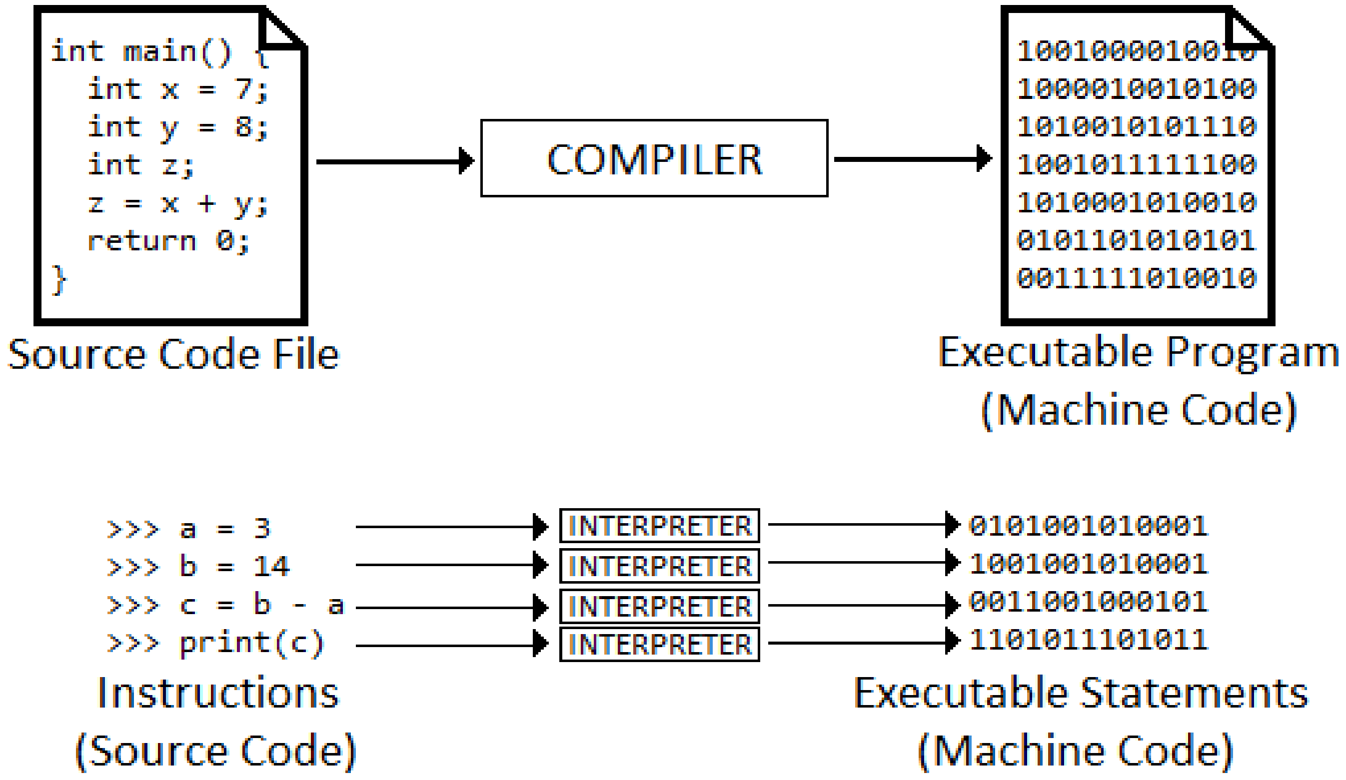


What is a high-level programming language?

- A ***high-level programming language*** is one where the instructions read more closely to a human language.
  - Normally, the language will work for a variety of different platforms/processors.

- Programs in high-level languages are easier to read, write and update when compared to programs written in a low-level language.

Compilers and Interpreters



How are programming languages classified?

- Most programming languages follow a *paradigm* or style of how instructions are written.
- The two most common types are:
  - ***Procedural Programming*** which seeks to break computer programs into separate routines or *procedures* that are sent to the processor to be executed.
  - ***Object-Oriented Programming*** which seeks to break computer programs into self-contained objects that use fields and methods to manipulate the program's data.
- Some languages, like Python and Java, are multi-paradigm.

What is Python?

- Python is a high-level, object-oriented programming language.

- A widely used general purpose language.
  - Normally ranked as a "Top 5" language in terms of popularity.
  - Can be used to create application or system software.

Who makes/made Python?

- Created by Guido van Rossum of the Netherlands.
  - Started development in 1989 as a hobby.
  - Released in 1991.

- Maintained and developed by the Python Software Foundation.
  - Non-profit organization devoted to Python’s continued development.

- van Rossum remains the principal author of Python.

How does Python work?

- Python source code is written by a programmer.
  - ***Source code*** is the human-readable text file written in a programming language that contains executable statements and instructions.

- The Python code is interpreted to machine code by the Python interpreter.

- Python code *can* be compiled, but it isn’t required.

What are the different versions of Python?

- Python 2
  - Last release (version 2.7) was in 2010.
  - Still widely used as not all of Python 2's libraries have been updated for Python 3.

- Python 3
  - Initially released in 2008; Current version (3.7.1)
  - Not all of Python 2's libraries are included yet.

What do I need to develop Python programs?

- In order to run Python programs, you'll need to have the Python interpreter installed on your computer.

- The interpreter can be used in two ways:
  - Interactive mode: You type Python statements and they are executed.
  - Script mode: You write a source code file containing Python statements. Then, the interpreter reads the source code and interprets the statements contained in it.

- Python scripts can be written using a simple text editor, like Notepad.
  - Better tools exist to simplify the development process.

Creating a Python program

- Python 3 comes with IDLE
  - **I**ntegrated **D**evelopment and **L**earning **E**nvironment
  - Allows writing and executing Python source code files.
- Appropriate for new Python developers.
- More powerful IDEs exist for developing Python applications.

Creating a Python Program

- Statements entered in interactive mode are not saved to a file.
  - Interactive mode is useful for testing and learning new Python statements.
- We will typically want to save our statements to run our program again.
- Python scripts (source code files) are saved in a text file with the .py filename extension.

Characteristics of a Modern Programming Language

- Keywords
- Operators
- Programmer-Defined Names
- Syntax

Keywords

- A ***keyword*** (or ***reserved word***) is word that has special meaning to a programming language.
  - The word is *reserved* from being used in other contexts within programs written in the language.
  - Keywords are typically used in a language for performing some specific process.
- For example, in many languages the word “if” is a reserved word.
  - The if keyword begins a special statement that allows a program to make a decision.
  - **if** *this* then do *that*
- Many different languages utilize the same keywords.

Operators

- An ***operator*** is (usually) a symbol that performs an operation on one or more *operands*(values/data).
- In the mathematical expression 1 + 2, the plus sign is an operator that adds the two operands together.
  - In this example:
    - **1** and **2** are operands
    - **+** is the operator
    - **Addition** is the overall operation performed by the operator.
- Many languages use the same operators for performing common operations, like arithmetic and comparisons.
- In some cases, keywords can take the form of an operator.

Punctuation

- ***Punctuation*** is characters or symbols used when writing statements in a programming languages.
  - A *statement* is like a sentence or an instruction in a programming language.

- Consider the sentence *I went to the park, the mall, and the college.*
  - We used punctuation for listing multiple places (commas) and a period to end the sentence.
  - Programming languages will use characters in similar ways.
    - For example, commas are often in used programming languages when specifying a list of values.

- Punctuation varies among different languages.
  - Some languages, like Java and C require ending statements with semicolons.
  - Languages like Python do not require punctuation at the end of statements.

Programmer Defined Names

- A ***programmer defined name*** is an identifier, whose name is created by the programmer.

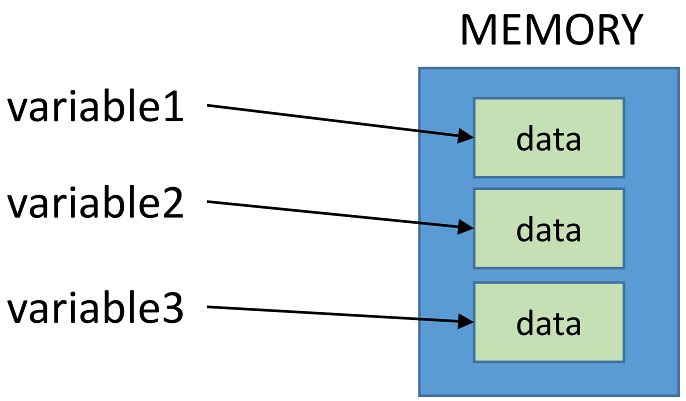
- A programmer usually chooses a name that describes the data or value it represents.
  - For example, an identifier we name “height” would probably represent a numeric measurement.
  - An identifier we name “city” would probably represent the name of a city or town.

- These identifiers (and the names we choose for them) help us to keep track of how we use data in our programs.

Variables

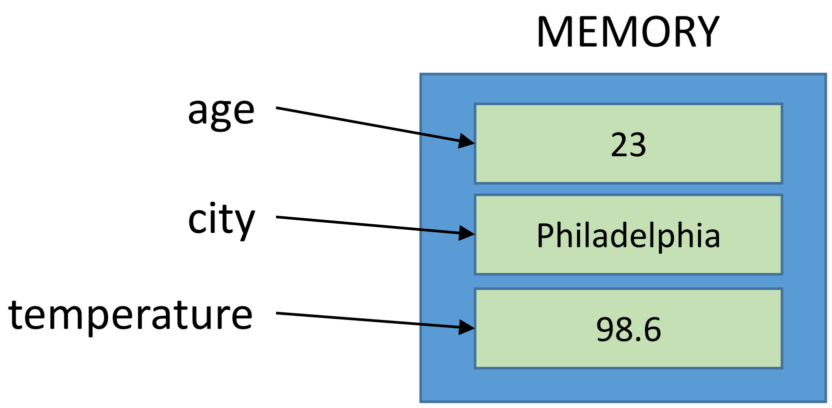
- A ***variable*** is a type of identifier that represents a location in memory where data is stored.

- Like the name suggests, the data referenced by a variable may vary.
  - New values/data can be assigned to the memory location the variable references.



Variables

- Variable names are programmer defined.
  - We choose variable names based on the data they represent in our programs.



Syntax

- ***Syntax*** is the language's rules for how keywords, operators, punctuation, and identifiers must be arranged in statements.

- The rules for how statements are written are paramount.
  - It ensures the instructions of a program are correctly executed.

- "Tall, he is."
  - We can kind-of understand what this English statement is saying.
  - If unclear, a computer can't "guess" as to our intentions when we give it instructions.
    - A statement is syntactically correct, or it is not. There can be no ambiguity.

Syntax

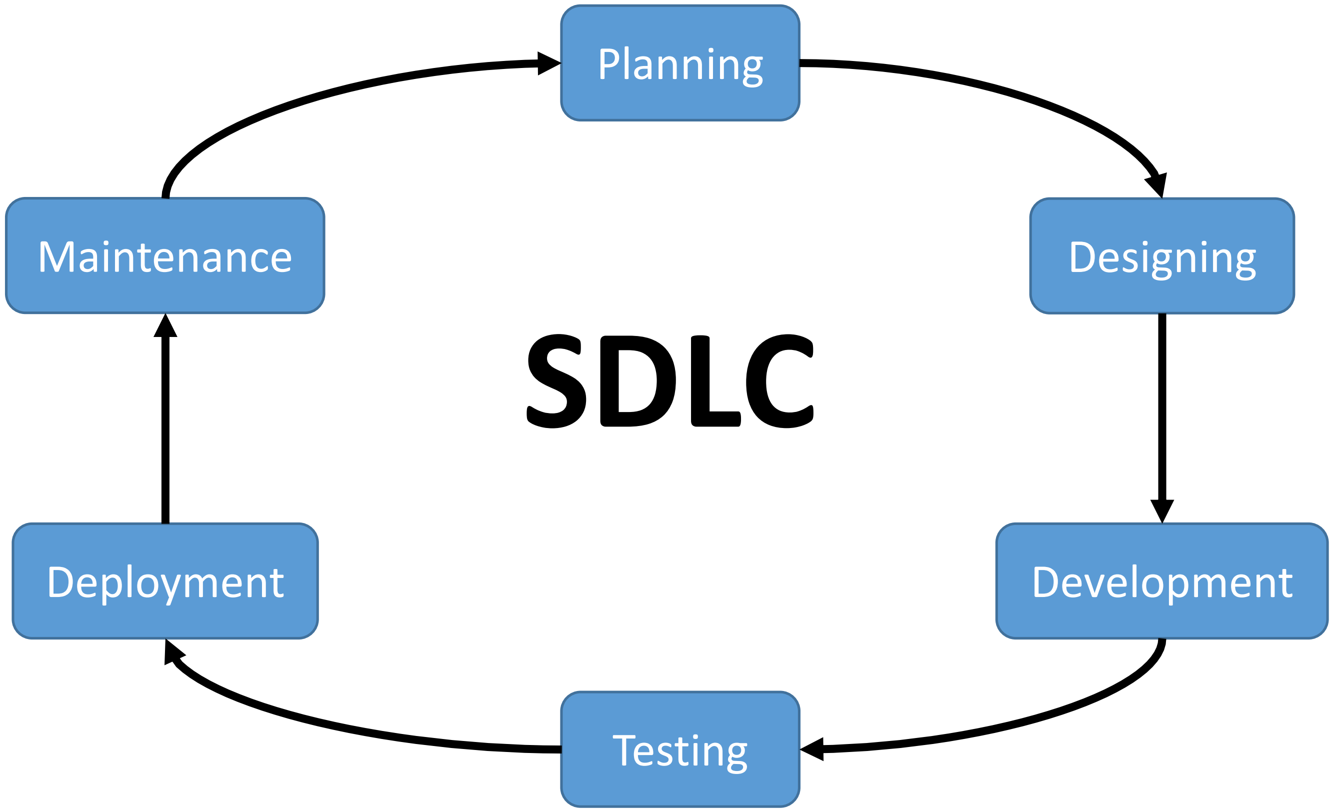
- A language's syntax is usually the most notable difference among different programming languages.
  - How languages accomplish tasks is usually comparable, but how we write those statements to accomplish the task typically differs.
- Some languages have comparable syntax.
  - Many languages are derived from or inspired by other languages.
  - Python shares some similarities with other languages, but overall has many differences in syntax.

The Software Development Life Cycle

- The ***Software Development Life Cycle*** (SDLC) is a process to produce computer software.
  - Highest Quality
  - Lowest Cost
  - Shortest Time

- Consists of (normally) six stages.

The Software Development Life Cycle



SDLC Stage 1 – Planning

- The Planning Stage involves input from all project stakeholders to determine the project's objective.
  - The Customer
  - Senior Management
  - Sales/Marketing
  - Technical Experts

- This is also when an estimate of resources and costs is determined.
  - Equipment, labor, etc.

SDLC Stage 1 – Planning

- The Planning Stage is sometimes called a ***requirement analysis***.
  - What do we want?
  - What don't we want?

- Risks
  - Is the project's timeline feasible?
  - Does the technology exist?
  - Is the cost too high?
  - **Minimize Risk**

SDLC Stage 1 – Planning

- Near the end of the Planning Stage, the requirements of the product will need to be formalized.
- A **Software Requirement Specification** (SRS) document will outline what functionality the product should have.
  - Requirements should not be ambiguous.
    - “Good User Interface”
  - This document should be reviewed and approved by stakeholders.

SDLC Stage 2 – Designing

- The Designing Stage involves creating the overall architecture of the application.

- **Design Document Specification** (DDS) documents will contain different design approaches for the architecture.
    - Is based on the SRS
    - With input from stakeholders, the best design approach is selected.

- Each approach should:
    - Identify the separate components of the architecture.
    - Identify how the components will work together.
    - Ensure the application's requirements are met.

SDLC Stage 2 – Designing

- The DDS should also contain a list of milestones
  - What will be completed in certain timeframes?
- Functionality of the application should be detailed.
  - User Interfaces
  - Failure
  - Limitations
- Misunderstandings will cause problems later.

SDLC Stage 3 – Development

- With the requirement analysis and design document complete, software development can begin.
  - The better requirements were defined in the previous stages, the easier it will be for the programmers to create the actual product.

SDLC Stage 4 – Testing

- After development is complete, the product needs to be tested.

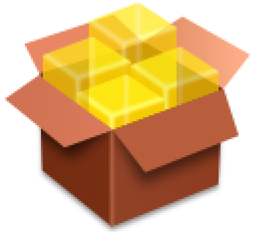
- While testing is performed by programmers as they develop, a formal test procedure or test plan must be created.
  - The test plan should incorporate testing the features and functions described in the DDS.

SDLC Stage 4 – Testing

- Some organizations have entire departments (***Quality Assurance*** or ***QA***) devoted to testing.
- QA testers follow the test plans to ensure the product works as intended.
  - Programming teams are notified if the testers discover issues.
- QA testers will also try to find and report any odd or abnormal behavior (*glitches*) in the product/application.

SDLC Stage 5 – Deployment

- After the product has passed all tests and is determined to function as designed, the product is ready to be delivered to the customer.



- Often, the deployment stage will involve teams who visit the customer on-site to install and configure hardware/software.
  - Will work closely with the customer's IT staff.
  - Ensures the product was delivered and is working correctly.

SDLC Stage 6 – Maintenance

- Problems may arise after deployment.
  - Issues not anticipated or discovered during testing.

- The customer will often be provided with an update or ***software patch*** that fixes the problem.

- Customer Support services may be offered.
  - Product support may have ***end-of-life*** terms.

What next?

- If this was a one-time software solution, the product and SDLC is complete.

- Normally, this isn't the end.
  - After getting customer feedback and patching problems, work for the next version of the software can be started.
  - The cycle begins again at the Planning Stage.

Developing Software

- During the Development stage (Stage 3) of the SDLC, the programming team will begin by reviewing and understanding the DDS.
  - Sometimes, this is the responsibility of a software development manager.

- Different parts of the application will be assigned to different team members.
  - Usually matched with their ability/expertise.

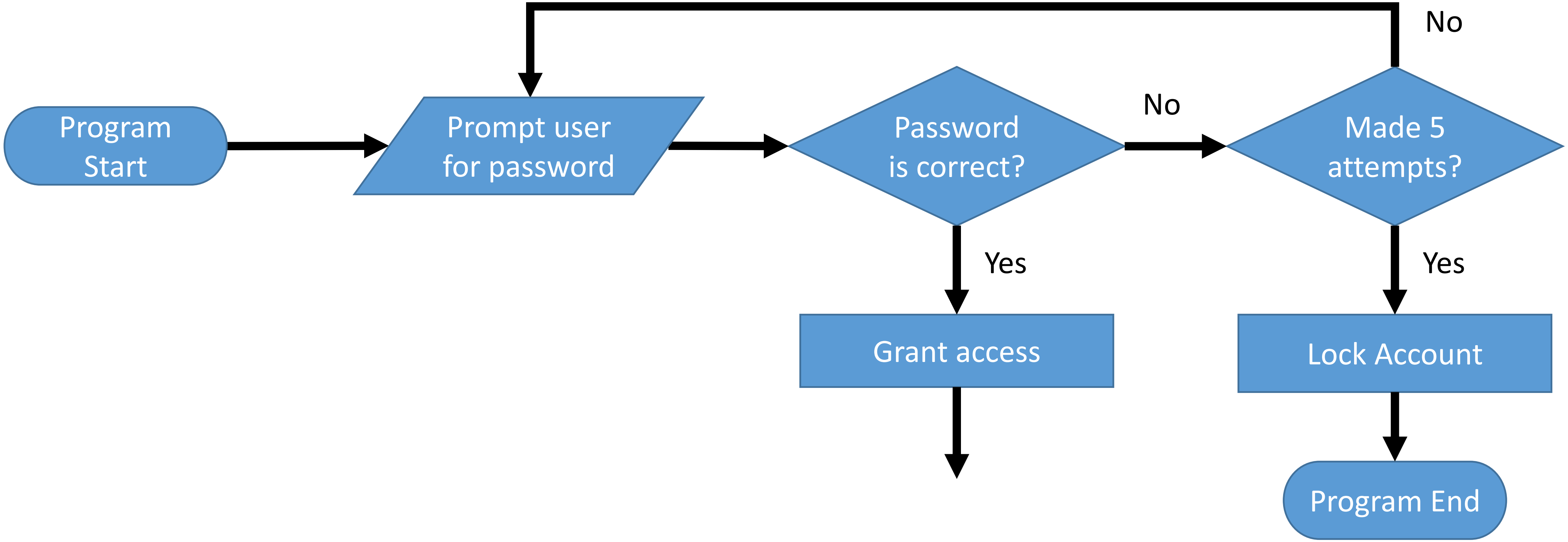
Developing Software

- Programmers use a variety of non-programming techniques when developing software.
- A programmer must have a plan **before** they write a single line of code.

"Plans are worthless, but planning is everything."
- Dwight Eisenhower

Developing Software – Flowcharts

- Drawing flowcharts is a great way to aid in your planning by visualizing processes.

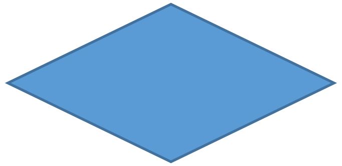


Developing Software – Flowchart Symbols

- Oval – Program start or stop

- Rectangle – Process

- Diamond – Decision



- Input/Output – Parallelogram

- Arrows – Direction of Flow

Developing Software – Pseudocode

- Based on the programmer's notes and flowcharts, a "script" of how the program should work can be written.
  - The script will contain the step-by-step processes completed by the program.
  - The processes are often written in plain text, mixed with actual programming code.

- This is referred to as ***pseudocode***.
  - It's not really valid, working code; Serves as a guide for how the actual code will be written.

```
Ask user for password
while user_guess != password :
    Print error message
    attempts += 1
    Check if too many attempts
        Print error message / Stop program
    Ask for password again
...
```

Developing Software – Programming

- With the completed flowcharts and pseudocode scripts, the programmer can begin to write the actual program.
  - You’ve already drawn/written out exactly (or pretty close, at least) how the program should function.
  - The flowcharts and pseudocode act as a road map of all the steps the program needs to take to complete its task.

```
Ask user for password
while user_guess != password :
    Print error message
    attempts += 1
    Check if too many attempts
        Print error message / Stop program
    Ask for password again
...
```

```
user_guess = input("Enter password: ")
while user_guess != password :
    print("Invalid Password.")
    attempts += 1
    if attempts == 5 :
        print("Login attempts exceeded.")
        exit()
    user_guess = input("Try Again: ")
```

Developing Software – Documentation

- Programmers will document their code using ***comments***.
  - Comments are notes that explains the "why's" and "what's" of the code.

- Other programmers may not understand what certain statements are doing, why they are there, and/or why they are important.
  - YOU might even forget why you have certain statements in the program.

- Properly documented code makes debugging, maintenance, and working as a team easier.
  - It also shows me that you understand what your statements are doing and why you wrote those statements.

Developing Software – Testing

- As the programmer develops the program, he or she must test that the functionality works correctly.
  - Many programmers will develop iteratively- create or change code and test it, create or change more code and test it, and so on.

- A programmer may encounter a few types of errors during the development process.
  - A ***compile-time error*** is an error that occurs when the program is compiled into machine code.

Developing Software – Testing

- A ***run-time error*** is an error that occurs while the program is running, causing the program to crash.
  - When a program crashes, the program will stop executing its statements.
- The source of a run-time error can sometimes be difficult to pinpoint and can require considerable time to solve.
  - When a run-time error occurs, it will often provide some details to help track down the cause.

Developing Software – Testing

- You may, during testing, discover your program exhibit unintentional behaviors or glitches.

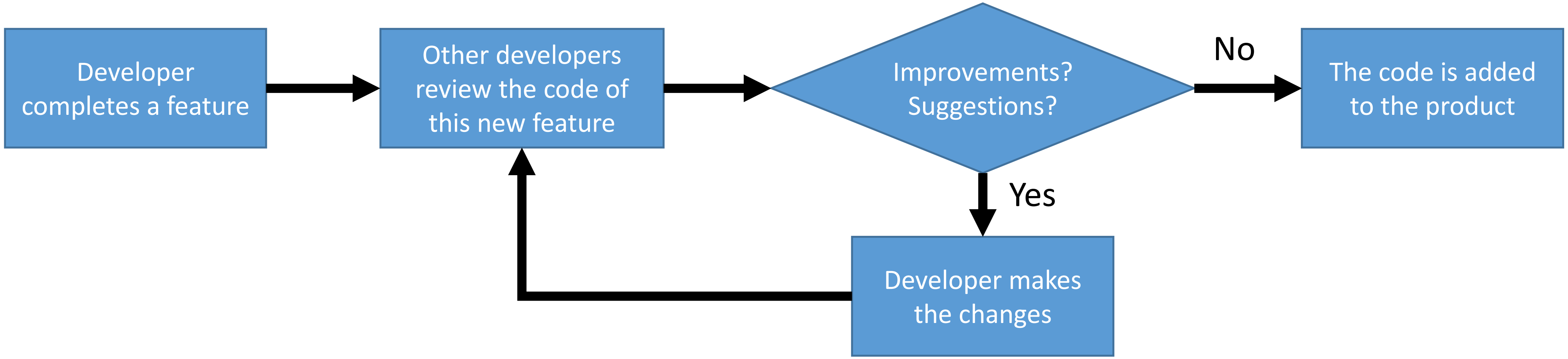
- A ***bug*** is a colloquial term for some erroneous code, logic, or unexpected behavior in a program.
  - ***Debugging*** is the term used to describe the process of searching for the cause of an error or unexpected behaviors.

Developing Software – Best Practices

- Always start a program with a pencil and paper.
  - Draw flowcharts
  - Write a pseudocode script.
- Test, Test, Test.
  - Validate your program works as designed and there are no bugs.
- Manage your time effectively.
  - Expect to spend time planning, programming, and testing/debugging.

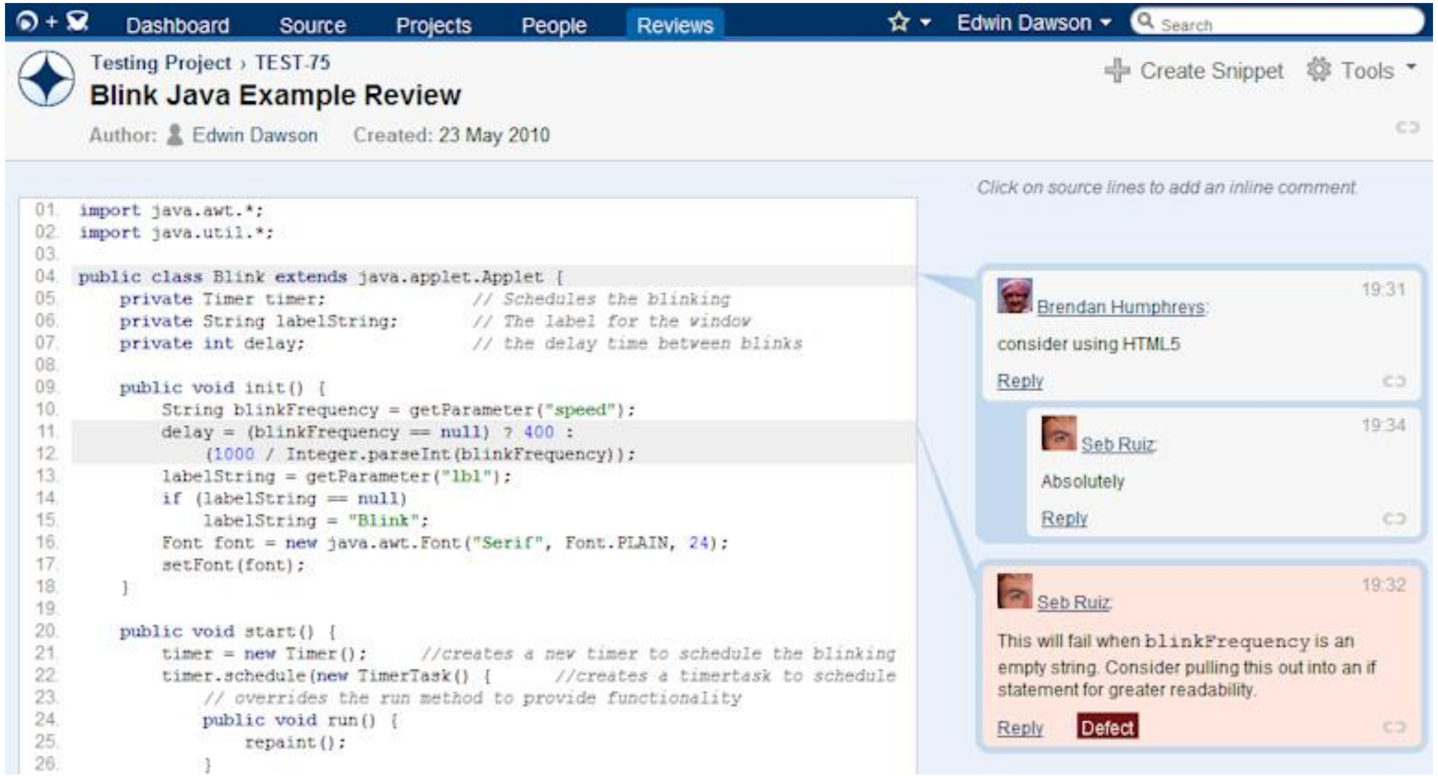
Developing Software – Code Reviews

- A ***code review*** is when developers meet to look over each others code.
  - Normally involves the more experienced and senior developers.



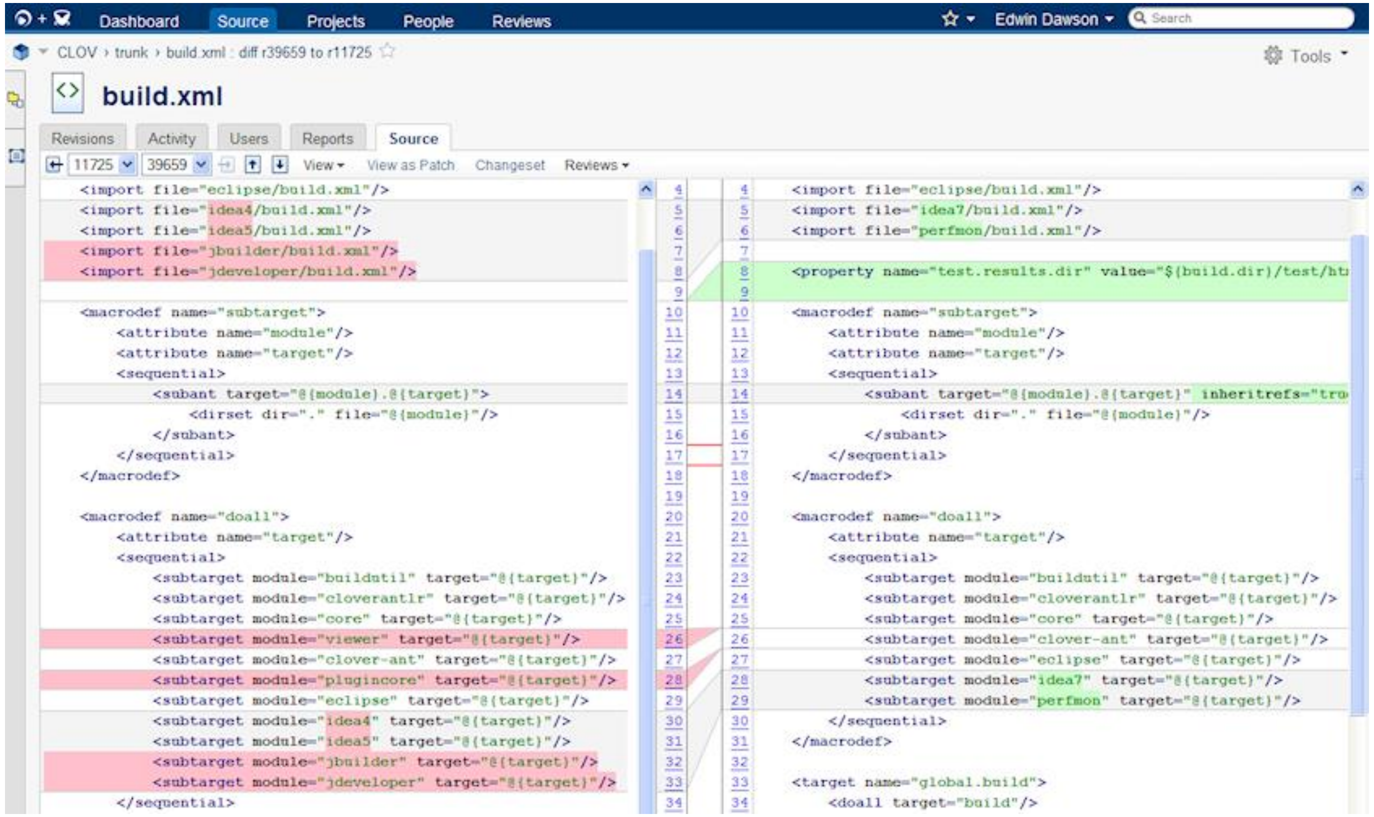
Developing Software – Code Reviews

- Crucible is a popular (paid) tool used for code reviews.



Developing Software – Code Reviews

- FishEye is another popular (paid) tool used in code reviews.



Developing Software – Code Reviews

- A code review also allows other developers to become familiar with what functionality/features are being added.

- Code reviews let all developers on the team better understand how the different components of the application are working.

Developing Software – Issue Tracking

- There are a number of tools like Jira (paid) and Bugzilla (free) available for tracking issues, code changes, and project management.