Basic Programming with Python

Prepared By:

Professor Dr. Md. Mijanur Rahman
Department of Computer Science & Engineering
Jatiya Kabi Kazi Nazrul Islam University
www.mijanrahman.com

1

Computer Programming

CONTENTS

1.1. COMPUTER PROGRAMMING

Computer programming is the process of designing and building software applications, tools, and systems. It involves writing code in a programming language, testing and debugging it, and then integrating it into a larger software system. Programming languages are used to write computer programs. Many different programming languages are available, each with its own syntax, features, and limitations, and popularity can vary depending on the field and industry. However, here are some of the most popular programming languages used today:

- **Python:** A high-level, versatile language that is popular for data science, web development, and automation.
- **Java:** A versatile, object-oriented language that is popular for building large-scale applications, particularly in enterprise environments.
- **JavaScript:** A scripting language that is widely used for building interactive web applications and mobile applications.
- **C++:** A high-performance language that is popular for systems programming, game development, and other performance-critical applications.
- **C#:** A modern, object-oriented language developed by Microsoft that is used for building Windows desktop applications, games, and web applications.
- **PHP:** A popular server-side scripting language that is used for building dynamic websites and web applications.
- **Swift:** A language developed by Apple for building iOS, iPadOS, and macOS applications.
- **Kotlin:** A modern language developed by JetBrains for building Android applications and is also used for server-side development.
- **Ruby:** A high-level, dynamically typed language that is popular for web development and automation.
- **Go**: A language developed by Google for building highly concurrent systems, network servers, and other performance-critical applications.

Computer programming (or software development) typically involves several stages, as shown in Figure 1.1. The major stages are summarized below:

1. **Problem statement:** Identifying and analyzing the problem that the software is intended to solve.
2. **Requirements gathering:** Understanding the requirements of the software system and what it needs to accomplish.
3. **Design:** Creating a high-level design of the software system, including the algorithms, data structures, and overall architecture.
4. **Coding:** Writing the actual code in a programming language, following the high-level design of the system.
5. **Testing:** Debugging the code and ensuring that it works correctly through the use of test cases and other techniques.
6. **Documentation:** Creating and maintaining documents that describe the software system, its architecture, design, functionality, and other important aspects.
7. **Maintenance:** Updating and fixing the code as needed over time to keep the software system running smoothly.

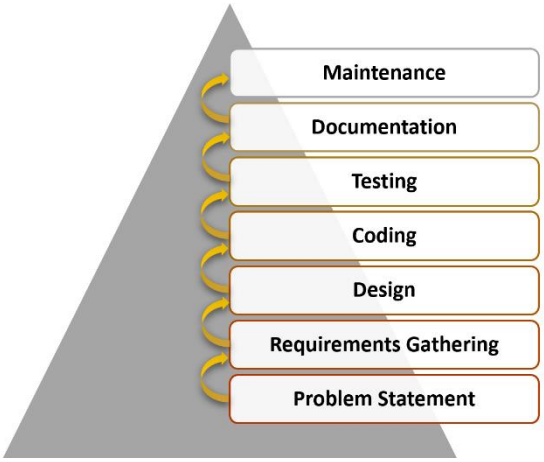

Figure 1.1: Stages in computer programming (or software development).

Programmers:

Programmers may work on a wide range of software projects, from small scripts to automated tasks to large-scale enterprise systems. They may work on projects as individuals or as part of a team, and they must have strong problem-solving skills and be able to work effectively with others. Computer programming is a rapidly evolving field, with new programming languages, tools, and techniques being developed all the time. Programmers must continually learn and stay up-to-date with these changes in order to remain effective in their careers.

1.1.1. Problem Statement

A problem statement in software development is a clear, concise, and specific description of the problem or challenge the software intends to solve. The problem statement should comprehensively understand the problem, including its root causes, impact on stakeholders, and relevant constraints.

The following are the key components of a problem statement in software development:

1. **The problem:** A clear and concise description of the problem or challenge that the software is intended to solve. It should include details such as what the problem is, why it is important to solve it, and how it affects stakeholders.
2. **The context:** A description of the context in which the problem exists, including any relevant industry trends, regulatory requirements, or market factors.
3. **The impact:** An explanation of the impact of the problem on stakeholders, including the negative consequences of not solving the problem.
4. **The constraints:** Any relevant constraints or limitations that affect the solution to the problem, including technical, financial, and organizational constraints.
5. **The objectives:** A list of specific goals that the software solution should achieve, including both functional and non-functional requirements.

By defining the problem statement clearly, software development teams can ensure that they develop software solutions that address the problem's root causes, meet stakeholders' requirements and expectations, and deliver value to the organization. It also helps to prioritize tasks, allocate resources effectively, and manage project risks.

1.1.2. Requirements Gathering

Requirements gathering is an essential step in the software development lifecycle. It involves gathering, analyzing, and documenting the needs and expectations of stakeholders for a software project. The requirements-gathering process is typically done in the project's early stages, laying the foundation for the entire development process.

The requirements-gathering process can be broken down into several steps:

1. **Identify stakeholders:** The first step in requirements gathering is to identify all the stakeholders who will be involved in the software development process. This includes end-users, project managers, developers, and other stakeholders who have a vested interest in the project.
2. **Define the problem:** Once the stakeholders have been identified, the next step is to define the problem that the software will solve. This involves identifying the business needs, the pain points of the end-users, and any other issues that the software needs to address.
3. **Gather requirements:** The next step is to gather the requirements for the software. This can be done through interviews, surveys, workshops, and other methods. The goal is to capture as much information as possible about what the software needs to do and how it needs to do it.
4. **Analyze requirements:** Once the requirements have been gathered, they need to be analyzed to ensure they are complete, accurate, and feasible. This involves reviewing the requirements for clarity, consistency, and completeness and identifying any conflicts or ambiguities.
5. **Prioritize requirements:** Once the requirements have been analyzed, they need to be prioritized. This involves identifying which requirements are essential and which can be deferred to later stages of the development process.
6. **Document requirements:** The final step in requirements gathering is to document the requirements. This involves creating a requirements document that outlines all the requirements for the software, including their priority level, any constraints, and any other relevant information.

The requirements-gathering step is critical to the success of a software development project. It ensures that all stakeholders are on the same page regarding what the software needs to do and how it needs to do it. By following a structured requirements-gathering process, developers can create software that meets the needs of end-users while staying within the constraints of the project.

1.1.3. Design

The design stage in computer programming is an essential step in the software development lifecycle. It involves taking the requirements gathered during the requirements gathering stage and creating a plan for how the software will be built. The goal of the design stage is to create a blueprint for the software that developers can use during the implementation stage. The design stage can be broken down into several steps:

1. **Define the architecture:** The first step in the design stage is to define the architecture of the software. This involves identifying the components of the software and how they will interact with each other. The architecture should be designed to meet the requirements gathered during the requirements gathering stage.
2. **Create a design document:** Once the architecture has been defined, the next step is to create a design document. This document outlines the details of how the software will be built, including the data structures, algorithms, and interfaces that will be used.

3. **Choose a programming language:** During the design stage, the programming language to be used to build the software is chosen. This is based on factors such as the requirements of the project, the expertise of the development team, and the available resources.
4. **Create a prototype:** Creating a prototype is an optional step in the design stage. A prototype is a basic version of the software that can be used to test the design and get feedback from stakeholders.
5. **Review and revise:** Once the design document has been created, it should be reviewed by stakeholders, including developers and end-users. Any feedback received should be used to revise the design document to ensure that it accurately reflects the project's requirements.
6. **Create a detailed plan:** The final step in the design stage is to create a detailed implementation plan. This plan should include timelines, resource allocation, and any other relevant details.

Moreover, it ensures that the software is designed to meet the requirements of the project and that developers have a clear plan for how to implement the software. By following a structured design process, developers can create efficient, scalable software that meets the needs of end-users.

1.1.4. Coding

The coding stage in computer programming is the third stage in the software development lifecycle. It is the stage where the design document created in the previous stage is transformed into actual code by the development team. During the coding stage, developers write the source code for the software using the programming language chosen in the design stage. The coding stage can be broken down into several steps:

1. **Set up the development environment:** The first step in the coding stage is to set up the development environment. This involves installing and configuring the necessary software tools, such as text editors, compilers, and debuggers.
2. **Write the code:** The next step is to write the code for the software. This involves translating the design document into actual source code using the chosen programming language. The code should be well-organized and documented to ensure that it is easy to read and maintain.
3. **Test the code:** Once the code has been written, it should be tested to ensure that it is functioning as expected. This involves running tests to check for bugs, errors, and other issues. Any issues found during testing should be fixed before moving on to the next step.
4. **Debug the code:** Debugging is the process of identifying and fixing issues in the code. During the coding stage, developers should use debuggers and other tools to identify and fix any bugs or errors in the code.
5. **Refactor the code:** Refactoring is the process of restructuring the code to improve its readability, maintainability, and performance. During the coding stage, developers should continuously refactor the code to ensure that it is optimized for the requirements of the project.
6. **Review and revise:** Once the code has been written and tested, it should be reviewed by the development team and other stakeholders. Any feedback received should be used to revise the code to ensure that it meets the requirements of the project.

Thus, the coding stage involves writing the source code for the software using the programming language chosen in the design stage. By following a structured coding process, developers can create efficient, reliable software that meets the needs of end-users.

1.1.5. Testing

The testing stage in computer programming is an essential step in the software development lifecycle. It involves testing the software to ensure that it functions as expected and meets the project's requirements. The goal of the testing stage is to identify and fix any bugs or errors in the software before it is released to end-users. The testing stage can be broken down into several steps:

1. **Test planning:** The first step in the testing stage is to create a test plan. This plan outlines the types of tests that will be conducted, the tools and resources used, and the criteria for determining whether the software has passed or failed the tests.
2. **Test design:** Once the test plan has been created, the next step is to design the tests. It involves identifying the various scenarios the software will encounter and creating test cases to simulate them. The test cases should cover all aspects of the software, including its functionality, usability, and performance.
3. **Test execution:** The next step is to execute the tests. It involves running the test cases and recording the results. Any issues encountered during testing should be documented and reported to the development team.
4. **Defect tracking:** During the testing stage, defects or issues identified during testing should be tracked using defect tracking tools. This helps to ensure that all defects are addressed and resolved before the software is released to end-users.
5. **Test reporting:** Once testing is complete, a test report should be generated. This report summarizes the results of the testing and provides recommendations for improving the software.
6. **Retesting:** After the defects have been addressed, the software should be retested to ensure that the issues have been resolved and that the software is functioning as expected.

Hence, the testing stage is critical in the software development lifecycle. It ensures that the software is tested thoroughly and that any issues are addressed before the software is released to end-users. By following a structured testing process, developers can create reliable, efficient software that meets the needs of end-users.

1.1.6. Documentation

Documentation is an essential part of software development. It is the process of creating and maintaining documents that describe the software system, its architecture, design, functionality, and other important aspects. Documentation is crucial for software development projects as it helps developers, project managers, and stakeholders to understand the system, its behavior, and its components. The following are the key types of documentation in software development:

1. **Technical documentation:** This type provides detailed information about the software system's technical aspects, such as its architecture, design, coding conventions, data structures, and algorithms.
2. **User documentation:** This type of documentation is intended for end-users and provides information on how to use the software, including instructions, tutorials, and user manuals.
3. **System documentation:** This type describes the entire system, its components, and their interactions, including hardware and software configurations, network infrastructure, and third-party software integrations.
4. **Requirements documentation:** This type defines the requirements for the software system, including functional and non-functional requirements, use cases, and system constraints.

5. **Test documentation:** This type describes the testing process, including test cases, test plans, and test reports.

Documentation is critical for software development projects as it enables developers and stakeholders to understand the system and its components, communicate effectively, and maintain the system over time. It also helps manage project scope, track progress, and ensure compliance with regulatory and quality standards.

1.1.7. Maintenance

The maintenance stage in computer programming is the final stage in the software development lifecycle. It involves making changes to the software to ensure that it remains functional and up-to-date after it has been released to end-users. The goal of the maintenance stage is to fix any issues that may arise and to improve the software based on feedback from end-users. The maintenance stage can be broken down into several steps:

1. **Bug fixing:** The first step in the maintenance stage is to fix any bugs or errors that may be discovered in the software after it has been released to end-users. This involves identifying the cause of the issue and making the necessary changes to the software to fix it.
2. **Upgrades and updates:** Over time, the software may need to be upgraded or updated to add new features or functionality or to fix security issues. During the maintenance stage, the development team should release updates and upgrades to the software to ensure that it remains up-to-date.
3. **Performance tuning:** As the software is used by end-users, performance issues may arise. During the maintenance stage, the development team should identify and address performance issues to ensure that the software runs smoothly and efficiently.
4. **Technical support:** During the maintenance stage, the development team should provide technical support to end-users who encounter issues with the software. This may involve providing documentation or tutorials, answering questions, or resolving issues.
5. **User feedback:** The development team should listen to feedback from end-users and use this feedback to improve the software. This may involve adding new features or functionality, improving the user interface, or fixing issues that end-users have reported.

Here, it ensures that the software remains functional and up-to-date after it has been released to end-users. By following a structured maintenance process, the development team can ensure that the software is reliable, efficient, and meets the end-user's needs over time.

1.2. PROGRAMMING LANGUAGES

Programming languages are formal languages designed to be used by computer programmers to write software programs. They are used to provide instructions to computers, and they allow programmers to express algorithms, data structures, and other complex computations in a way that can be understood by both humans and computers. Various types of programming languages are available today, such as C/C++, Java, Python, C#, SQL, PHP, HTML, Prolog, Lisp, Ruby, etc.

Programming languages can be divided into several categories, including:

1. **Procedural programming languages:** These languages use a procedural approach to programming, where a series of steps or procedures are defined to solve a problem. Examples of procedural programming languages include C, Pascal, and Fortran.
2. **Object-oriented programming languages:** These languages use an object-oriented approach to programming, where data is organized into objects, and the behavior of these objects is defined by the methods that they contain. Examples of object-oriented programming languages include Java, Python, and Ruby.
3. **Functional programming languages:** These languages use a functional approach to programming, where functions are first-class citizens and program execution is based on evaluating mathematical functions. Examples of functional programming languages include Haskell, Lisp, and Scheme.
4. **Scripting languages:** These languages are typically interpreted rather than compiled and are used for scripting and automating tasks. Examples of scripting languages include Perl, JavaScript, and Ruby.
5. **Logic Programming languages:** Logic programming is a type of programming paradigm based on formal logic. It is used to specify relationships between objects and to define rules that can be used to deduce new information from existing information. The most well-known logic programming language is Prolog (Programming in Logic).
6. **Low-level programming languages:** These languages are closer to machine language and provide direct control over computer hardware. Examples of low-level programming languages include Assembly and C.
7. **High-level programming languages:** These languages provide a high-level abstraction from computer hardware and are designed to be easier for humans to read and write. Examples of high-level programming languages include Python, Java, and Ruby.

Each programming language has its syntax, semantics, and features, and the choice of programming language depends on the specific requirements of the software being developed, including its intended platform, performance requirements, and target audience. Programmers can often work with multiple programming languages, and they often choose the best language for each project based on the specific requirements and constraints of the project.

1.3. PROGRAMMING (OR DEVELOPMENT) TOOLS

Programming or development tools are software applications that help developers create, test, debug, and maintain software applications. These tools can range from simple text editors to complex integrated development environments (IDEs) and offer a variety of features to streamline the software development process. These tools and applications can be grouped into several categories, including:

- **Text editors:** Text editors are simple tools that allow developers to create and modify code in a text-based format. Examples include Notepad++, Sublime Text, and Visual Studio Code.
- **Integrated Development Environments (IDEs):** An IDE is a software application that provides a comprehensive set of tools for software development, including a code editor, a compiler or interpreter, a debugger, and often a visual design environment. Some popular IDEs include Visual Studio, Eclipse, and Xcode.
- **Code Editors:** A code editor is a text editor specifically designed for software development, and it provides features such as syntax highlighting code completion, and version control. Some popular code editors include Visual Studio Code, Sublime Text, and Atom.
- **Compilers and Interpreters:** A compiler is a program that converts source code into machine code and is used to create executable applications. On the other hand, an interpreter executes code line-by-line, often used for scripting and dynamically typed programming languages such as Python.
- **Debuggers:** A debugger is a tool that helps developers identify and fix errors in their code, and it provides features such as breakpoints, stack tracing, and variable inspection. Debuggers can be integrated into IDEs or used as standalone applications.
- **Source Control and Version Control Systems:** Source control and version control systems are tools that allow developers to manage and track changes to their code, and they are often used in team-based development environments. Some popular version control systems include Git, SVN, and Mercurial.
- **Build Automation and Continuous Integration Tools:** Build automation and continuous integration tools are used to automate the process of building and deploying software, and they help ensure that code changes are tested and deployed consistently and efficiently. Some popular build automation tools include Jenkins, Travis CI, and CircleCI.
- **Testing and Quality Assurance Tools:** Testing and quality assurance tools are used to validate the functionality and performance of software, and they help ensure that software meets the requirements of users and stakeholders. Some popular testing tools include JUnit, TestNG, and Selenium.

These programming and development tools are critical components of software development, and they play a key role in enabling developers to create high-quality software that is reliable, scalable, and secure. By leveraging these tools, developers can streamline the software development process, improve productivity, and enhance collaboration within development teams.

⌘⌘⌘