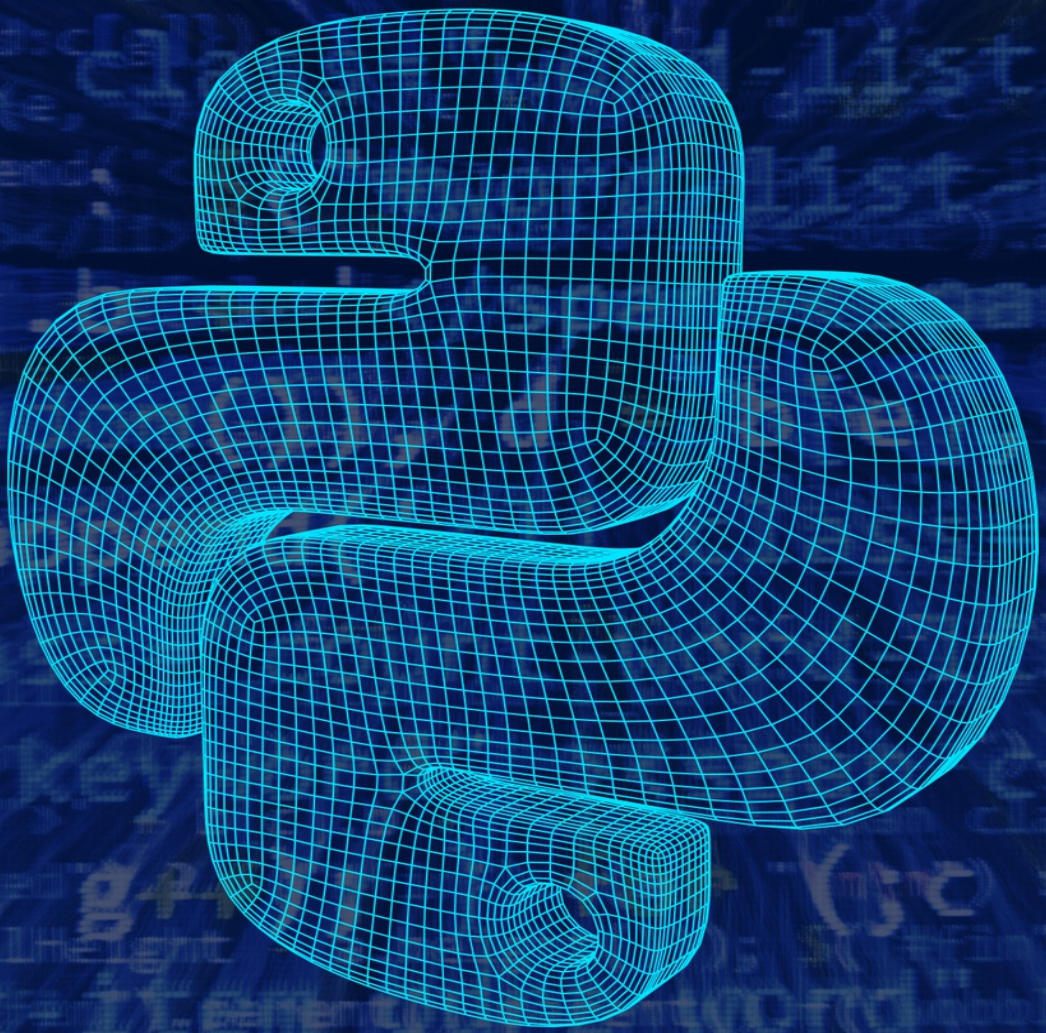
PYTHON PROGRAMMING FOR BEGINNERS



A CRASH COURSE TO LEARN PYTHON AND OTHER RECOMMENDED CODING LANGUAGES IN USE TODAY

STEVE GEDDIS

Python Programming for Beginners

A Crash Course to Learn Python and Other Recommended Coding Languages in use today

Steve Geddis

Table of Contents

Introduction

Congratulations on purchasing *Python Programming for Beginners*. By reading this book, you will discover some particularly important information that will help you as you begin your journey to becoming a proficient computer programmer.

The first chapter is an in-depth dive into the foundation of programming and coding. The chapter begins with an introduction to programming and continues to explain the features and differences between low-level languages and high-level languages. The chapter continues to trace the origin of computers back to the 1800s and explains the changes over the years to where we are today and where we expect to be soon.

The chapter continues to explain the features of the different computer languages in terms of generation, explains the reasons it is so important for someone to study programming, and concludes with the relationship between coding and programming.

Without programs, computers would not be anything close to what we have today. The second chapter contains a comprehensive list of the most popular programs we have and use today together with their functions. The chapter explains 12 of the most common programs we have, and the value the end-user derives from the use of such.

One of the reasons computers are fast becoming man's best friend is because it is possible to communicate with them on so many levels. Chapter three is an investigation into the computer languages that a beginner should start to learn and understand for him or her to begin programming.

These languages help translate human communication into a form that a computer can understand and process some feedback. The chapter concludes with a recommendation of the easiest programming language beginners should learn.

Computer languages out there are so many, and since each language has its own features and advantages over others, chapter four explores the most common coding languages out there and some of the reasons why developers love them so much.

The chapter goes into detail explaining the best languages for front-end development and the best for back-end development. The chapter concludes with an explanation of how a developer can narrow down to a coding

language depending on the specific project he or she wants to undertake.

The fifth chapter is about the fundamentals and techniques each coding language requires. This is important to know because many of these languages have a lot in common and many processes in one can be like the next. The sixth chapter is about the best practices that coders and developers abide by when coming up with codes, software, or even programs. Learning these best practices will help a beginner have a better understanding of the development field.

The seventh chapter is about compiling codes, and it explains the role of a compiler and goes into detail explaining its workings. The final chapter is a rich resource for absolute beginners who want to learn to code within the shortest possible time, and it concludes with several projects such a beginner should consider starting with.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy it!

Chapter 1: What is Programming?

In the world today, people use languages such as Spanish, English, and Mandarin, etc. to communicate with one another. However, when it comes to computers, communication and languages are unique and different from human languages. How do people communicate with computers since computers cannot understand human language or any other form of human communication?

Programming is the process of taking an algorithm and encoding it into an idea referred to as a programming language, which a computer can easily understand and execute. A programming language refers to the high-level and unique languages that a programmer utilizes to instruct computers to perform specific tasks. He or she creates scripts or software programs and uses them to command the computer to carry out specific actions.

Computer programming is necessary in the contemporary world because people use it in running the systems for nearly all devices we have today. Programming languages enable people to command machines, and they obey.

Machines and humans have quite different ways of thinking and processing information, but programming languages now act as the bridge between humans and computer communications. These have facilitated the interaction between computers and humans and made the relationship remarkably effective.

Computer programming has changed the world since the growth and development of computers over the past few decades. Today, people can communicate with computers without much difficulty because technology has improved the relationship so much that people can practically speak to some computer programs and get the correct feedback in their own languages. Computer languages have played a huge role in enhancing this relationship, and it continues to grow with every new advancement in technology.

Fundamentally, programmers created two major kinds of programming languages, namely High-Level Languages and Low-Level Languages.

Low-Level Languages

Low-Level Languages are basic computer instructions, which also go by the name of machine codes. A computer is not able to understand any form of instruction given by a user in human language; however, it can easily

understand codes, which after processing the codes can give the proper responses to the users.

The main purpose of low-level languages is to intermingle with the hardware of a computer system. They play the role of assisting in managing, operating, and synchronizing all the system components and the hardware of the computer. Low-Level Languages handle every instruction that forms the architecture of the hardware systems.

- Machine Language

One of the main languages discovered by programmers suitable for machines is the basic low-level languages. The creators of this language created it with the initial intention of interacting with the initial or first-generation computers. This language is written in machine code or binary code. Binary code means that the coding system comprises of two digits, 1 and 0. One (1) and zero (0) represent a character, digit, or a letter in machine language.

Binary code underlines every language on a computer. This indicates that numeric values are stored in a computer using only two main digits, 0 and 1. In binary, the zeros and ones represent “ON” and “OFF” in that order. This demonstrates the representation of numbers in a computer device.

- Assembly Language

This is the second invention of programming language. This is the development of a machine language where numbers are not necessary. In this programming language, English words, symbols as well as names are used. This is the most essential computer language for processors.

Advantages of low-level languages

- Any program that is translated needs less memory
- Overall control over the code
- The code written in these languages can be executed faster
- This language can work directly on the storage sites
- The language can make use of hardware or particular machine-dependent instructions such as specific chips

High-Level Languages

A high-level language is the kind of language that facilitates the growth of a

program that is friendly to the user in a complex programming background. These languages are close to the human language and enable the programmer to focus on the issue that needs a solution. This kind of language mainly focuses on the programming logic instead of underlying hardware elements like register utilization or memory.

The programming style of a high-level language is simple to implement and learn compared to low-level languages. The complete high-level language code focuses on the particular program a developer needs to form. This language does not need any addressing of the hardware constraints when programmers are developing it.

However, every program written in this language ought to be interpreted into a machine language before it is performed or executed by a computer. The knowledge of any hardware is not necessary for high-level languages, because the language is not attached to the computer. Examples of high-level languages include C++, Pascal, Python, Java, and Visual Basic.

One of the important features of high-level languages is that it gives programmers the chance to write programs for every type of computer and system. Every instruction in a high-level language should change to machine language for the system to understand it.

- Scripting languages

Scripts are also known as scripting languages. This kind of language uses high-level construction that allows it to construe as well as perform one command at a time. These languages are simple to study compared to the compiled languages. Some major examples of script languages include JavaScript, Pearl, and AppleScript, etc.

- Object-Oriented Languages

This type of high-level languages mainly focuses on “objects” instead of the “actions”. To attain this, the focus is mainly on data rather than logic. The main reason behind this is that the programmer cares about the object he or she desires to manipulate instead of the logic required in them. Some of the object-oriented languages include C+, Java, C++, Python, and Swift, etc.

- Procedural Programming Language

This form of a programming language is well structured and comprise of

difficult procedures in its programming to create a complete program. This kind of language has an efficient order of functions as well as commands to finish a task or a program. Examples of procedural programming languages include FORTRAN, BASIC, ALGOL, and COBOL.

Advantages of High-Level Languages

- Faster to write code because it uses languages like English
- The code is manageable which means that it is not considered to run in a single machine
- It is easier to debug in developments as a result of the English like statements
- Users can modify the English like statements

The History of Programming

During the early days of programming, programmers had not discovered the high-level languages. However, users had to write their code at the machine or simply assembly language level. These meant that users had to have a lot of knowledge of the underlying machine without any abstraction over the functionality of the processor. However, this resulted in numerous issues.

During these days, users could not separate data and code. In most cases, the code would overwrite the program during its execution. Conversely, the control structures were quite simple statements. However, in a complex program, it was hard to understand the compilation of a program.

A woman named Ada Lovelace established the first computer program in 1883. She developed an algorithm for running a particular ancient analytical engine. This engine was the property of Charles Babbage created to calculate the Bernoulli's numbers. The algorithm enabled the success of this calculation accurately. As a result, most people consider this algorithm as the first programming language in existence. However, the growth and development of technology in the modern world have influenced the creation of more programming languages.

- **1800 – Jacquard Loom**

A man named Joseph Marie Jacquard taught a loom how to read and examine punch cards. This was the first complex multi-threaded processing unit. Silk-weavers greatly opposed this kind of innovation because they were worried

that the robots would take over their occupation.

- **1842 – Algorithm for Analytical Engine**

Ada Lovelace was a mathematician who created the first Algorithm for the Analytical Engine for Charles Babbage to compute Bernoulli numerical. People consider this the first programming language ever created. People believe that Ada translated an Italian book to English and learned how analytical engine works. In her theory, she created a language for the machine that never existed.

- **1936 – The Bombe**

Alan Turing, a mathematician by profession, worked as a top-secret agent in breaking military codes used by Axis powers and German authorities. With the assistance of his coding team, Turing was able to crack the secret of coding by creating a computing machine that was capable of breaking codes.

- **1949 – Assembly Language**

This programming language was in use in the Electronic Calculator originally. The language used in this calculator was the Low-Level programming language. The low-level language was used to simplify the language of the machine code. This was the particular way of giving the computer instruction on what to execute. An example of a device that made use of this assembly language is the EDSAC calculator.

- **1952 – Autocode**

This general term was used for the family unit of the ancient computer programming languages. Alick Glennie in the United Kingdom invented the primary one for Mark 1 Computer at Manchester University. Several people believe that Autocode was among the first, if not the first, computer programming language to be compiled. The advantage of Autocode was that a compiler program could successfully translate it into machine code directly.

- **1957–Fortran (Formula Translation)**

Mr. John Backus was the creative mind behind the development of the FORTRAN language. This programming language was applicable in complex mathematical and technical tasks such as accurate statistical

analysis. FORTRAN is an acronym derived from the term Formula Translation, and hence, its full application in the scientific fields of work. This is the oldest language and used computer-programming languages. Today, it is in use in different tasks.

- **1958 – Algol (Algorithmic Language)**

Another language developed for scientific applications and tasks was Algol. In addition, an acronym, Algol, represents the phrase Algorithmic Language. Algol formed the basis of future programming languages at the time. Programming Languages such as C++, Java, C, and Pascal developed later from the foundations laid by Algol. This language was the product of a group of scientific individuals in the year 1958.

- **1959 – COBOL (Common Business Oriented Language)**

The Common Business Oriented Language, also known as COBOL, was the creation of Dr. Grace M. Hopper. All the kinds of computers at the time from various brands could run COBOL. Its application included in government computers, telephone systems, credit card finance, health facilities, and ATMs. COBOL entered into infrastructural deployment with traffic lights and automobile systems using this language as well. COBOL entered the market in 1959.

- **1959 – LISP**

In addition, in 1959, Mr. John McCarthy developed the LISP programming language. This development happened at the Massachusetts Institute of Technology (MIT). The acronym LISP represents the term List Processing. As a basis for the future Python and Ruby languages, LISP application was in researching artificial intelligence (AI) in 1959.

- **1964 – BASIC (Beginner's All-purpose Symbolic Instruction Code)**

BASIC was a programming language created by Mr. John Kemeny and Mr. Thomas Kurtz. However, its initial development was for the non-technical computer science students at Dartmouth College. BASIC is an acronym as well. It targeted beginners from all backgrounds and contained elementary instructions that were easily understandable. Bill Gates and Paul Allan modified the new description of BASIC and developed the earliest Microsoft

Product.

- **1970 – Pascal**

Niklaus Wirth developed this program. It was given the name Pascal in honor of a French mathematician, philosopher, and physicist Blaise Pascal. The program is easy to learn. The creator made it as a tool to teach computer programming. This computer program was the main language used in the development of Apple software in the early years.

- **1972 – Smalltalk**

Adele Goldberg, Dan Ingalls, and Alan Kay created this program. The programming language permitted computer programmers to amend code on the take-off. This programming language was also useful in the introduction of other factors now current in ordinary computer programming languages comprising Java, Ruby, and Python.

- **1972 - C**

Dennis Ritchie developed this computer programming language at Bell Labs. Many people recognize C as the earliest high-level language. This language is similar to human language although it has gone through so much editing to suit machine code. This programming language was invented for an operating system known as UNIX, which runs on many computers. The C code has inclined other languages together with Go, C#, Ruby, Java, Perl, PHP, JavaScript, and Python.

- **1972 – SQL (Structured Query Language)**

Raymond Boyce and Donald Chamberlin at IBM developed this programming language. This language, known as SQL, made requests via software queries, which enabled the user to view or modify information stored in the database. SQL is an acronym for Structured Query Language. The language was used for viewing as well as changing information that is stored in databases. In addition, the programming language relies on if command sentences known as queries to remove, add or view data.

- **1978 – MATLAB (Matrix Laboratory)**

The development of a language called MATLAB happened next during the

seventies. This language, whose acronym represents Matrix Laboratory, was the creation of Mr. Cleve Moler. This language was especially useful in a research setting involving technical and mathematical tasks. It was the first language to allow for the realization of two and three-dimensional visual representations.

- **1983 – Objective-C**

The eighties were a decade that began with the development of the Objective C programming language. Mr. Brad Cox and Mr. Tom Love were the two innovators responsible for its creation. This language was common in manufacturing Apple products that carry operating systems programmed by Objective C.

- **1983 – C++**

The next development that followed was the creation of the C++ language. Mr. Bjarne Stroustrup created this language. A wide array of applications adopted C++, including Adobe Photoshop and the software used in many gaming systems. High-performance systems typically have a component of C++ in their program codes. This language development occurred in 1983.

- **1987 – Perl**

Larry Wall formally created Perl in 1987 as a scripting language that was planned for text control. The main intention of this computer programming language was to make the processing of reports much easier. Today, this program is widely used for different purposes including Web development, Linux system administration, and network programming.

- **1990 – Haskell**

This computer programming language was named after an American mathematician and logician Haskell Brooks Curry. This purely functional computing programming language essentially means that 90% of its use is on mathematics. It is used in numerous companies particularly the once that deal with complex records as well as computations.

- **1991 – Python**

The development of the Python programming language happened next. Mr.

Guido V. Rossum was responsible for its creation. Monty Python, a British comedy troupe, was the inspiration behind the development of Python language. Many people relate favorably to Python due to its ease of comprehension. It also needs a smaller number of lines of program code than most of the other languages. An example of a popular application that uses Python is Instagram. Python came about in the year 1991.

- **1991 – Visual Basic**

After Python, Visual Basic programming language followed hot on its heels. Visual Basic is a language that enables users to alter a substantial lump of program code easily. This language uses a graphic user interface (GUI) for interactions with the programmer. This GUI makes it easier for the user to develop his or her program code in a drag and drop manner. Visual basic arrived in the year 1991 as well.

- **1993 – R**

Robert Gentleman and Ross Ihaka at Auckland University, New Zealand designed this language. R language was named based on the initials of the two authors. This language is mostly employed in statistics to perform different types of data analysis.

- **1995 – Java**

Sun Microsystems designed this language primarily for hand-held devices and cable boxes but developers made further improvements to make it useful on the Internet. Java is common in computers and Android Phones as well as special meters.

- **1995 – PHP (Personal Home Page)**

A programming language used primarily for web development came into existence after Java. This language, known as PHP, was the creation of Mr. Rasmus Lerdorf. Its name is an acronym for Personal Home Page, and it often operates on web servers. PHP is a language that is currently common in websites and many online blogs. It also has an application in the creation of WordPress, which is, in turn, responsible for developing personal web pages and websites. The development of PHP happened in 1995.

- **1995 – Ruby**

Ruby was a programming language developed by Mr. Yukihiro Matsumoto. It is a multipurpose language with a broad range of computing tasks. However, it is often exclusive to the development of applications based on the web. Ruby language runs a slower program code than most of the other programming languages. This time allowance is usually enough for developers to write lines of program code as well as executing them. The creation of Ruby language also occurred in the year 1995.

- **1995 – JavaScript**

Another web-based language development occurred right after Ruby. This new language was JavaScript. Mr. Brendan Eich was responsible for its development in a short period of around ten days. Most websites currently use JavaScript since it improves user interactions with web browsers. Besides, this development happened in 1995.

- **2000 – C#**

Microsoft entered the new millennium with the development of a new programming language. This language was C#, and it incorporates the clarity of Visual Basic with the functionality of C++. Most of the features in this language are identical to the Java language. However, its functionality operates based on the C++ language. Microsoft used C# for the development of desktop applications contained in its operating system. Most other Microsoft devices also run on C#. This creation of C# occurred in the year 2000.

- **2003 – Scala**

Martin Odersky developed this. The Scala is the type of programming language that joins useful or practical programming with object-oriented programming. Object-oriented is prepared in the region of data that controls the right of entry to code. The compatibility of Scala and Java has contributed to the development of Android.

- **2003 – Groovy**

Another derivative of Java was the development of the Groovy language. This development, courtesy of Mr. James Strachan and Mr. Bob McWhirter, allowed for the skill improvement of its users. This improvement was due to its simplicity and precision when studying it. The creation of Groovy took

place in the year 2003.

- **2009 – Go**

Google came up with this programming language to tackle issues that often occur in big software systems. Since the use of technology and computers is different now compared to the past when Python and C++ had just been developed, issues started when people started using big computer programs. Go programming language was meant to enhance the working setting for programmers so that they can read, write, as well as maintain large software systems more effectively.

- **2014 – Swift**

Apple came up with this programming language because it wanted to replace the C, C++ as well as Objective-C. Swift is effective and simple to employ and permits little mistakes. It is adaptable and can be utilized for desktop as well as could services and mobile apps.

Computer Programming Languages Today

According to the history of computer programming languages, it is evident that the preceding computer programming languages inspire modern programming languages. Today, some of the old programming languages serve as a stable foundation for modern ones. However, newer computer programming languages are more effective and make programmers work easier.

Almost all the leading businesses in the modern world heavily rely on different computer programs in order to meet all their business needs such as transactions, data collection, analysis, and processing, as well as research to meet their client needs. For example, medicine, as well as science, requires accurate and complex programs in their research, also, phone apps ought to be updated to meet client demands. The new and mounting needs guarantee that programming languages remain to be significant.

Overview of Generation Languages

The categorization of programming languages based on their history results in the following generations:

I. **First-generation languages**

These languages occupy the bottom level in the hierarchy of programming

languages. This generation consists exclusively of machine language. No elements of human language exist at this level. 1GL is another way to refer to these languages.

II. Second-generation languages

These languages constitute assembly languages and algorithms. Another reference to this generation is 2GL. Common examples of 2GL include programming languages used in editing video content and executing video games. Computer hardware drivers and kernels also use 2GL.

III. Third-generation languages

These languages exist in the higher tiers of the programming language hierarchy. Another name for these languages is 3GL, and their examples include Java, JavaScript, C, C++, and Visual Basic languages.

IV. Fourth-generation languages

These languages are identical to the human language and often exist in database programs and their related scripts. Also known as 4GL, these languages include SQL, Python, Perl, Ruby, and PHP.

V. Fifth-generation languages

These languages enable the creation of programs with the aid of graphical features resulting in efficiency from the programmers due to the associated ease of program execution. 5GL is a reference to fifth-generation languages, as well. Examples of 5GL include Prolog and Mercury.

Reasons Programming is so Important

The development of a programming language involves a series of steps. A programmer normally describes the issue, devices a resolution, codes the program, puts the program into a test and records the program. In this case, the programmer illustrates what he or she understands and then selects a program to employ and debugs it in different stages to make certain that no errors are present. Finally, the programmer designs and tests it for use.

The ever-changing face of technology, programming is one of the exciting ones because it challenges the environment of new programmers' desires. However, programmers need to have a full understanding of how computers work. Today, more than a few universities, multiple professors and scores of

students use the programming language to complete tasks.

A significant aspect of programming is that it enables people to interact with computers and machines. Programming is also important because it harnesses the power of computing in all human venture of creating intelligent machines as well as tasks. Today, programming is significant for speeding up productivity in a machine. Programming is vital to collect, automate, calculate, and analyze the procedure of data in the machine.

Today, programming is useful in the creation of software as well as apps that assist computer systems and mobile phones to be useful in day-to-day life. It is important for people to understand and know how to use a programming language in their daily lives. Programming languages are the main reasons behind innovations in modern technology.

Today, programming languages are useful in robots, artificial intelligence, machine learning, the IoT (Internet of Things), blockchain technology, and cloud computing. Programming languages are the main path to creating software that carries out different or particular tasks in a systematic manner. A programming language is the language of communication with machines through a systematic format.

1. **System Knowledge**

Programmers have full knowledge of how and why computing works. The knowledge of programming language involves limitations, and this can set realistic probabilities and work surrounding the limitations to maximize the use of the equipment as well as accessories.

2. **Creative Platforms**

Programming highlights creativity particularly in providing solutions and in entertainment. Today, developers create games, graphics, video games, and animations using programming languages. In addition, they use it to introduce new business ideas and provide solutions to problems.

3. **Interactive Education**

In web development, programming allows new interactive web applications that have access to system resources and offer the same level of control as desktop applications. The interactive web applications are useful in online learning platforms. Today, the applications allow distance-learning programs

for students' impressions. Every institution comprises of a form of online learning implementation.

4. **Defining the Future**

Computer programming principles influence how modern technologies like voice-recognition and artificial intelligence work. The impacts of the programming languages in the modern world continue to influence technologies in the modern world. For instance, the trend toward computerized internet searches is more localized and proceeding. In the future, the hardware platforms developed will play a major role in computer technology.

5. **Machine learning**

Since computer systems operate with numbers, programming allows people to represent machine-like language in a human-readable format. This approach reduces debugging.

Relationship and Differences Between Programming and Coding

Programming and Coding are two vital strategies in Software Development Industries. Coding is the procedure of creating codes from one particular language to the other. Coding is a subject of programming because it equips first ladder programming. Coding engages inscription codes in diverse languages as initiated.

Programming is the process of mounting or performing machine programs at different levels by implementing them without any error. Programming is the procedure of writing codes so that human inputs, as well as corresponding machine outputs, remain in sync.

Relationship Linking Coding and Programming

Coding	**Programming**
The procedure of inscribing codes in different languages	The course of mounting or creating programs in a machine that executes a set of instructions.
Its basic objective is to ease communications linking humans and machines.	This is the course of inscription codes while ensuring human effort or input and machines stay in sync.

It is the first step in bringing in programming. Coders can have a reduced amount of proficiency than programmers	It is the fundamental communication linking human thinking and machine outputs comprising of compound structures. Programmers are more capable than coders are.
It is the original stage of programming which comprise of complex questions and it is simple in comparison to programming	Programming executes diverse difficult conditions and questions to come up with suitable machine level productivity. It is the advanced edition of coding as well as different strategies. It is more difficult compared to coding.
As an ordinary step of communication, coders usually deal with specific lines of codes without considering the details	Programmers handle communication plans in a developed way. They analyze and conceptualize diverse ways factors of communication as well as produce the correct machine productivity effectively.
There is different community support for the coders that help people to use dissimilar coding strategies as the existing engineering standards require	Programming is the factor of coding and several communities support it for continuous enhancement per the current standards.

The Key Differences Between Coding and Programming

- Coding is the procedure used to translate and write codes in languages while programming is the procedure of structuring an executable and utilizable program to conduct appropriate machine level outputs.
- Coding focuses specifically on the codes while programming is concerned with programs to be in command of as well as interact with machines to come up with accurate results.
- Coding is the process of interpreting the needs and the logic into

languages that computers/machines can understand while Programming deals with analyzing and developing the codes and engage the artifacts to make the system function properly.

- Coding involves programming strategy that includes interpreting needs, writing lines of codes, as well as putting into practice machine decipherable inputs, while programming deals with all processes from debugging to testing and execution. It handles the main functionalities between inputs and proper machine level productivity.

Chapter 2: Most Popular Programs and their Functions

Computer systems rely on programs to perform their various functions. From an end user's perspective, the most critical component of the system is the application software. This software is responsible for performing specific tasks tailored to the client's needs.

Currently, many businesses and personal activities depend on particular application software to carry out their operations successfully. As a result, this software has extensive usage by many people around the world. The most popular types of application software based on their associated functions include:

1. Word processing software

This application software enables the end-user to generate otherwise hand-written documents. It has features that allow it to hasten this process. Such features are in the form of tools that would enable the end-user to make necessary alterations in the document. These tools possess the following functionalities: blank page display on the user interface and entering text by typing. Editing and formatting the material are word processing tools as well. The software application also has an additional printing tool as a feature that makes physical printouts possible.

Word wrap is a feature common to all word processing software. This feature automatically moves the guide cursor to the subsequent line when the user finishes his or her current path. In combination with the appropriate dictionaries and thesaurus, editing becomes an effortless task for the user. You do not have to worry about spelling errors while typing your document continually.

Looking for spelling errors is typically a tedious chore for a human end-user. The software makes this task very simple and quick for you. The software has an inbuilt spelling checker and correction tool. This auto-correct feature connects to the appropriate language dictionary that corresponds to the user's input language. As a result, the auto-correct feature automatically makes corrections to any spelling errors.

The auto-complete function is a feature that performs tasks as specified by its name. It automatically completes a word for the user without having to type the whole word physically. This auto-completion task is possible through an algorithm based on predictive heuristics. This tool employs a feature that

depends on the probability of the most common words currently in use for a particular language. A typical example of software dedicated to word processing tasks is the MS Word from Microsoft Corporation. Other notable basic examples of less used text editors include the Notepad and WordPad.

2. Presentation software

Presentation software is used to relay information to an audience in a manner that is simple and easy to follow. It uses a unique graphic and visually appealing user interface as its platform onto which your raw information awaits further dissemination to your spectators. This software is a means of effective communication, especially in a busy office setting. Your colleagues do not have the time to go through a detailed report and would prefer a quick summary instead.

The software platform contains slides as a feature that allows for the smooth transition from one idea to the next. This feature is inherent in all the presentation software currently in the market. Your audience quickly grasps the information. Besides, the various aesthetics that applies to the presentation will fascinate them. The fascination raises interest from the listeners, which, in turn, leads to increased and closer attention to your message. As a result, effective communication occurs through this process. A famous type of presentation software is the MS PowerPoint from Microsoft Corporation.

3. Spreadsheet software

This software allows the user to manage his or her data in a format based on grid patterns. The user interface contains a box grid template within which data input occurs. Spreadsheet software is standard in the financial sector for usage in tasks that require a lot of accounting. The grids are adaptable to both text and numbers or any data needed by the user.

However, it is very efficient at handling various types of numerical data. Calculation features are inbuilt within the software, so you do not have to make mathematical computations manually. Most of the accounting processes are automatically doable. The mathematical formulas are flexible, depending on the calculation desired by the user.

Spreadsheets usually sort out, examine, and evaluate arithmetical data and

information. The results of the preliminary evaluation generate visual outputs that are understandable to humans. Such output information is often in the form of graphs and charts. Further analysis of this output gives the user vital information needed to make crucial decisions. Maps and graphical information easily envision the trend of a particular data set over a specified period.

Other information easily depicted includes the ranges of different sets of data. Therefore, spreadsheets are responsible for the conversion of raw numerical data into useful visual information. This resulting information forms the basis on which vital decisions apply. Two famous examples of application software that is great for spreadsheets include MS Excel from Microsoft Corporation and Apple Numbers.

4. Database management software

This term represents the digital storage format for all the raw data to which the database management software controls the access, edition, and any other form of manipulation of this stored data. It controls the creation and structure of relational databases, as well. Databases take the format of cells or boxes formed by the intersection of vertical columns and horizontal rows, and they contain the relevant data. This software is available in a variety of fields and environments that make use of substantial amounts of data. Such places include most educational institutions, health facilities, and large commercial businesses.

The most popular form of data storage structure currently in use is the relational database. Each cell within the database has an assigned specific area and kind of data with which to interact. The tools within the software enable the end-user to connect and communicate with the database. This communication takes the form of electronic instructions inbuilt within the tool's functionality.

These functions include form creation, data filtration, report authoring, and the definition of relevant criteria. The capacity of this software extends to information retrieval from different locations via reports and queries. The recovery of reporting forms containing the relevant information is possible as well.

During interaction with a specific relational database, the end-user requests information through queries. Any person can use this request to view, edit,

and exploit the relevant data to his or her needs. The resulting output takes the shape of the previously mentioned information reports and data forms in a summarized format as well. Examples of such database management software include Microsoft Access, Lotus Approach, Corel Paradox, and Oracle.

5. Internet browsing software

This category of application software links the end-user to the internet. Its platform acts as a window from which a person can gain access to the internet. These particular software types are known as internet browsers. The name represents the act of marauding through various websites online, looking for specific content. These internet browsers allow for the increased connectivity currently experienced around the world. A user can quickly form a link with another individual located at a far-flung place in an instant. Commercial activities are also present online.

Fierce market competition often necessitates the movement of most businesses to web-based platforms. Buying and selling goods or services have become electronically portable activities. For you to engage in online marketing and commerce, you will need a capable platform. Browsers provide this required platform. The two famous examples of internet browsers are Internet Explorer from Microsoft and Chrome from Google.

6. Multimedia software

This software enables the end-user to create and manage particular media content. The data subjected to multimedia software include videos, images, and audio content. The user can create and edit his or her video content if he or she has a camera. Any other device capable of capturing video information or data may serve as a replacement for the camera. The content from this initial data-capturing device faces a transfer to the multimedia software platform.

This platform contains the editing tools with which the user can manage his or her content. Once satisfied, the edited content can convey the relevant information that you initially intended. The same process applies to video content that pertains to audio content and images captured as photos. The software allows you to manipulate the original photo to an output image that meets your desired aim. Examples of application software that run multimedia content include Windows Media Player and Real Player.

7. Educational software

Educational software is a bulk of applications that makes studying easier for computer students. This software acts as a tutorial for the dissemination of knowledge through computers. In addition, any teacher may use the software as a tool for instructions to his or her computer students.

Lay people who are not necessarily involved in the technological world can use this software to gain general knowledge as well. Examples of the educational software include MATLAB, Google Earth, and Dictionaries, such as Britannica and Encarta.

8. Enterprise software

Enterprise software is standard in businesses that deal with prospective customers and consumers regularly. It is crucial to have this software when running a business that interacts with many shoppers who often buy your product. The software enables you, as an entrepreneur, to keep track of your buyers and their purchasing tendencies.

For instance, you may use this software as a platform for tailoring your products to individual shoppers. This software collects a significant amount of user data. This information is vital for future effective interactions between the salesperson and his or her customer. An example of enterprise software is the Customer relationship management system.

9. Information worker software

This software is standard in many large businesses. The software allows managers to keep track of the progress of different projects all from a single portal. In addition, it allows for the efficient management of the human resource components of the business. The business directors and leaders can keep track of the deficiencies and redundancies of the workforce.

Major decisions affecting workers in various departments and divisions depend on information from this software. For instance, as a human resource manager, you can allocate annual leaves based on data contained in the software. Hiring more people or letting go of redundant workers are decisions that depend on the software as well.

In addition, the platform enables the allocation of projects and tasks to various employees. It allows for the proper time and resource management

for specific projects. As mentioned previously, tracking the associated progress becomes simple. Every businessperson must keep track of employee needs and shortcomings to achieve success eventually. The software enables you to perform this vital management function seamlessly. Examples of information worker software include resource management tools. Documentation tools are another example, as well.

10. Engineering and product development software

This software may confuse with an application suite. An example that falls in this category is the Integrated Development Environment (IDE). An integrated package has limited features that are inherent to the individual software applications. This software is available in a single bundle capable of performing the roles of spreadsheets, word processors, presentation, and database management.

However, unlike an application suite, these roles and tasks found in IDE are minimal. Only the basic functionality of the relevant job is available in an IDE package. This platform is useful in cases that require project development by multiple developers from a collective point.

11. Simulation software

Simulation software is a specialized kind of program that applies to specific situations. Therefore, these software platforms are not as common as the other applications mentioned herein. They are exclusive to training facilities that deal with scientific research and exploration. Examples of simulation software include experimental and flight simulators.

For instance, trainee pilots practice their skills on flight simulators before involving themselves in real-world planes. The flight simulator is typically identical to the real cockpit found in aircraft. This software enables novices to perfect their skills without the associated harm associated with inexperience.

12. Application suites

An application suite is a bundle of multiple application software grouped as a single package. The various applications in the suite perform similar or closely related tasks in an integrated work or business environment. This single pack contains all the features for spreadsheets, database management, presentation software, and word processing. Besides, all the suite's contents

are developed and distributed by the same company.

Examples of application suites include the Microsoft Office Suite from Microsoft Corporation and the OpenOffice suite from Sun Microsystems. The Oracle Corporation later took over the development and distribution of the OpenOffice suite. The Microsoft Office suite comes from the Microsoft Corporation. It consists of the applications mentioned above that specifically originate from the Microsoft Company.

Chapter 3: Which Programming Language Should a Beginner Start With?

Learning a programming language enables a person to begin coding. So, which programming language should a beginner start learning? The programming languages offer varying features, which means that there is no straight and correct answer to this question.

Instead, an individual should evaluate his or her needs to determine which programming language he or she should start learning. He or she should also take into consideration aspects such as dedicated community and intuitive syntax to aid in choosing where to begin. There is a saying, "Start as you mean to go on", and it applies here. A person who identifies and learns the right language will eventually be ready to learn to program effectively and successfully.

Easiest Programming Languages for Beginners

Below are some of the simplest programming languages that first-time learners can study and put in use. Understanding how they function will help a person to determine which programming language to start learning.

1. **Python**

Python is a programming language that a person utilizes to form desktop and web applications. He or she uses it as a vital tool in areas regarding scientific computing, machine learning, and data mining. Python provides one of the easiest and best programming languages for those at a beginner level. It also enables users to convey the idea of interest fully while using fewer lines of code.

Guido van Rossum created this programming language in the 1980s as a free, as well as open-source language kind. Python is easy to employ because it is flexible, procedural, contains dynamic language and is oriented to object (OOP). Additionally, it also offers functional programming styles. In any talks concerning coding, many people mention Python very quickly due to its popularity among teachers and learners. Many schools in Europe and the United States commonly provide Python in the introduction phase during the courses involving programming languages.

Python is an ideal programming language for a beginner to learn because it is straightforward and offers some fun when writing. Beginners find the syntax of Python program to be less strange, not intimidating, and more comfortable

to process. The Python source code used in this programming utilizes only white space, which makes the source code appear similar from one project to another. Furthermore, Python provides the Pythonic approach to coding, which is essentially the right way to encrypt. Thus, correctly writing the Pythonic code allows the beginner learner to read and comprehend it easily.

Additionally, a person can find tutorials regarding Python programming language easily due to its popularity among users. Research and knowledge are fundamental to any learning process. Hence, readily accessible Python tutorials are a huge benefit for the beginners. Some tutorials help a person to have a sense and familiarity with the language. Likewise, some sites or courses provide more interactive methods to help with a further comprehension of the programming language.

Learning the Python programming language is useful because many applications and fields apply it in their development. It uses web frameworks like Django to enable developers to create and power different back-end features and applications. It also provides dictionaries and libraries such as SciPy, which a person uses to initiate a particular project. As a result, Python enables development to proceed rapidly.

Therefore, Python is an excellent programming language for a beginner to learn because it is free, easy to understand, and offers some benefit to its future application.

2. JavaScript

JavaScript is a programming language that an individual uses to develop webs, mobile applications as well as web apps. Most modern browsers run this programming language. It provides a cross-platform nature that makes it easy for a beginner to learn which, in turn, makes it a popular programming language. All web browsers support JavaScript and can provide scripting, along with server-side languages, such as when using Node.js.

A person mainly uses JavaScript in front-end development that offers scripting language on the client-side. It provides an excellent introductory programming language for beginners despite receiving fewer recommendations in comparison to Python and Ruby programming languages. This shortcoming is because JavaScript is a programming language that does not use typing. Consequently, a person discovers any errors associated during the runtime.

It is also important to note that beginners should not confuse JavaScript with Java. Java is a programming language, while JavaScript is primarily a scripting language for front-end development. JavaScript is vital in web development and learning; it allows the person to understand the basics of the programming language. In addition, it also prepares him or her for further education related to web development, as well as C programming language. It is because the syntax of the C programming is similar to that of the JavaScript programming language.

It provides a solid education for beginners because it has a range of applicability. JavaScript programming language involves dynamical-typing or is untyped, and it is also more relenting when compared to other kinds. It is cross-platform, and it enables a person to function without a compiler as it runs in his or her browser. This feature makes it a popular programming language among users and learners. A beginner can also learn using JavaScript to access other complex programming languages such as C++.

A beginner may face some setbacks linked with JavaScript such as issues concerning untyped features where they do not show errors until runtime. However, a beginner can carry out some appropriate research that will enable him or her to comprehend JavaScript easily. It offers various benefits like cross-platform compatibility, among others. These advantages make it valid as an introductory programming language as it is the language of the web.

3. Ruby

Ruby is a programming language that facilitates multiple paradigms such as imperative, functional, and object-oriented programming. As a result, Ruby is a very adaptable programming language that assists a learning beginner in several ways. A person employs the Ruby programming language when developing applications based on the web, mobile apps, and websites.

Ruby provides a syntax that people can read easily. It contains a general-purpose and dynamic language that a beginner not only finds readable but straightforward as well. This syntax enables a person to learn many things, and use the programming without necessarily studying other numerous jargon and commands. An individual employs the Ruby programming language when dealing with the back-end development of his or her work.

Ruby on Rails refers to a web framework based on Ruby programming language. It comprises a group of Ruby shortcuts. A beginner can learn Ruby

on Rails and use it to develop web applications, such as a web blog in a matter of minutes. Learning Ruby and its associated framework like the Ruby on Rails can help a person in several ways. It can enable him or her to gain knowledge and skills in a commonly used programming language. It also helps him or her to get jobs because even the beginner level of expertise is highly valuable to various startups.

4. Java

Java is one of the most dependable and influential programming languages around. A person uses Java programming language to develop native applications on Android. He or she can also employ it as a server-side language. James Gosling developed it in the 1990s, and Oracle currently maintains it. Thus, Oracle's Java is among the oldest programming languages that exist.

Java is a programming language that is class-based and object-oriented. Therefore, it allows for the application of cross-platform and portability. Java usually is available during later or advanced stages in school because it utilizes syntax derived from other programming languages like C and C++. Nevertheless, it is also helpful to a beginner because it enables them to develop a thinking process similar to that of a programmer.

Even though Java builds many of its syntaxes on C or C++, it is still necessary for a beginner to know it. He or she learns and comprehends information processing in a PC. In turn, this knowledge provides a strong foundation for further coding learning and career. The analytical programming knowledge that a beginner gains from learning Java helps them to use it and address any problems connecting to Java. Moreover, many questions or challenges relating to Java already have answers or solutions since it is one of the oldest programming languages.

Hence, the beginners learning Java may need to first start by learning about C or C++ and develop their analytical thinking skills. Despite this setback, Java offers many benefits to the learner because it is a standard programming language with many resources. It provides long-standing language and a solid programming foundation, upon which the beginner can start and grow his or her programming skills. He or she can then effectively utilize Java in the subsequent application and web development.

5. PHP

PHP is a programming language that an individual utilizes to maintain scripts and develop websites. It is the most common server-side programming language, and it provides server-side scripting. Additionally, it can also create graphics and templates, create desktop operations and command-line script. PHP programming language offers a code that makes it easy for a learner to envision how it functions.

As a result, PHP is ideal for a beginner because he or she can perceive the code functionality use the programming language's versatility. PHP can apply to any significant operating system and enable the use of procedural or/and object-oriented programming. It helps a beginner to learn and understand the simple aspects of the programming language. It also assists him or her to access wide-ranging programming aspects, which he or she can apply in other complicated processes.

6. C and C++

C and C++ programming languages developed many parts of game engines, mobile apps, games, desktop apps, software, and operating systems. The C programming language helps to define the basics because it is among the essential programming languages in the field of computer science. Hence, a beginner will likely initially learn the C and C++ programming languages before moving on to others.

C programming language utilizes complicated code in comparison to others but still makes the basics clear. For this reason, a beginner needs to learn it despite the complexity of coding that it provides. C++ bases upon the C programming language and is one of the most critical programming languages around. It includes enhancement whereby, it enables object-oriented programming. It also allows a person to understand how things and information work in computer science.

A beginner mainly concerned with game development can start by learning C++ programming language because it features a lot in the development of games and game engines. C++ may be slightly more challenging to learn than C, but it offers many rewards to those who successfully overcome it. Thus, a beginner learning C and C++ programming languages gain valuable background knowledge of computer science. He or she also learns how to apply C and C++ in software and game development.

7. C#

C# is a programming language developed by Microsoft that a person uses in web and application development. It is excellent for beginners because it provides an Integrated Drive Electronics (IDE) that is not only simple to use but is also wide-ranging. The syntax in C# bases upon the C programming language. Hence, it helps the learner to gain a solid foundation in computer science since C provides primary programming languages of the field.

C# contains multiple and complex compilers and interpreters which may be a setback for first-time learners. However, the easy to utilize IDE, computer science background, along with the use of Visual Studio make C# ideal for a beginner to learn and use. Visual Studio simplifies the startup process and offers features like auto-generated files and auto-complete.

Beginners interested in video game developing can learn C# because it applies in this field heavily. It makes development more comfortable and faster when compared to previous languages in Microsoft. Basic knowledge of C and C++ programming languages assist a person to learn, understand and apply C#.

In conclusion, choosing which type of programming language to start learning depends on the needs and conditions of the learner. The above are the easiest ones that provide beginners with the essential basic knowledge that can further develop and advance their programming and computer science abilities.

Chapter 4: What are the Most Common Coding Languages?

The very first thing people new to the field of coding think about is where to begin. There are hundreds of different coding languages to choose from, each with its own idiosyncrasies and complexities. However, discovering the one most suitable for them, their interests, and their career goals can be quite difficult.

The easiest way to identify the best programming languages to learn at any given period is by looking at where the trend is going or listening to what the market is saying. Fortunately, as newcomers begin their journey as programmers, they will start to discover which coding language will most suit their objectives.

Before choosing the programming language to use, programmers need to consider several factors, such as:

1. The projects they are working on
2. Their background in logic and mathematics that might help the learn
3. Whether they want to work for an established company, go freelance, or work with a startup
4. If they are interested in website development, they need to determine whether they prefer working on the back end or front end
5. Whether they want to learn higher-end coding languages that are more flexible and have certain abstract programming concepts or focus on lower-end coding languages that have less abstraction and are closer to the hardware

Essentially, well-informed programmers can use the latest coding language trends and advances to their advantage by being economical about their learning. There are hundreds of programming languages out there and it seems like some super-geeks are creating new ones every week. This is why geeks are the new royalty in the modern world.

Anyway, this fast-paced evolution of the software development world is a great thing. Some of the best coding languages are also ideal for a wide range of popular applications, such as game development, mobile app development, website development and more.

Entrants into the world of software development need to go in with their eyes wide open to identify the coding languages that can boost their careers in the immediate future and beyond. They need to begin by learning about the most popular and powerful coding languages and then narrow down the list by focusing on the ones that make the most sense to their employment status, knowledge level, and other reasonable criteria.

Most Popular Back-End and Front-End Programming Languages

For people who are unfamiliar with software engineering and development, concepts like back-end versus front-end development can seem like a completely new language. Add all the coding languages, libraries, APIs, and frameworks that coders use to create software programs, and things can seem overwhelming.

Fortunately, newcomers in this field do not need to write code to grasp the basics of these concepts. However, it is important to understand one's career goals at the end of the software design boot camp to help determine whether one wants to focus on front-end development or back-end development. This can be quite tricky if one is not familiar with these terms.

Front-End Programming Languages

Front-end programming deals with everything end-users see as they are using or browsing the program or application. Programmers who handle front-end development are responsible for the feel and look of the application or site. Essentially, front-end developers focus on the client-side of the application.

They deal with things such as design, code analysis, debugging the program, as well as ensuring a seamless user experience. Front-end developers design the interface users visually see.

Some of the most common front-end coding languages include JavaScript, CSS, and HTML. JQuery is another front-end programming language, but it is going out of style. Prospective software developers should not be surprised to learn about legacy projects in their curriculum. This is because they still use the JavaScript library. They will also learn a lot about responsive design, color theory, grid system, and typography.

When new front-end programmers are thinking about the type of projects they will be working on, they should consider creating and redesigning sites. It is important to note that front-end programmers do not need back-end programming skills; however, having them is a significant advantage.

Websites developed by front-end programmers, for instance, do not need to interact with data stored on a database to function. Front-end content is constant, which means that large amounts of new data do not upload to affect the code. Restaurants and small businesses, for example, are good examples of front-end development or static sites.

- **HTML**

In the world of coding, things fall between two main categories, i.e., processes happening in the background and things users can visually see and interact with as end-users of a mobile app or visitors of a site. HTML, for instance, is one of the best front-end coding technologies or languages.

This coding language stands for HyperText Markup Language. Developers use this coding language to design the front-end part of web pages using what they call markup language, which defines the text documentation within the tag that defines web page structures. On the other hand, hypertext is the connection between different web pages.

This sounds somewhat basic, and that is because it is; actually, HTML is one of the most basic coding languages in the website development process. This is why it is so important to learn about its concepts and processes. To expand on its definition, consider the following:

1. HTML describes a webpage by telling browsers where to place videos, graphics, images, and paragraphs, as long as the header and footer.
2. When it comes to website development, HTML is the universally accepted standard. The fact that it is highly searchable makes it ideal for search engines such as Google, Bing, and Yahoo in terms of relevant keywords and search terms.
3. All web browsers, such as Google Chrome, Firefox, and Safari, understand this industry-standard language.
4. Popular browsers analyze HTML code and translate it into what end users see when they visit and browse websites.

Developers cannot get far building a website without a searchable and structured webpage and website. Therefore, HTML is one of the first coding languages developers should learn if they have a serious interest in coding.

Contrary to what new entrants to tech believe, HTML is not such a huge

undertaking; therefore, they need not panic. People who want to learn it should think about weeks, rather than years. However, it is better to spend a few weeks learning about HTML, in addition to CSS, its sister language used to define page layout styles, background colors, and fonts.

Developers who learn about these two coding languages will be in a better position to find paid work, such as building a custom email template or website or working as a social media administrator or manager. Even those who want to do developer-specific or higher-level programming work can still benefit from learning about HTML.

Adding HTML to their resume can have a positive effect on their income or salary at any job. However, those who decide to pursue higher-level, front-end website development should add to their skill set by learning about other coding languages such as JavaScript. However, as any experienced website developer would testify, HTML is the first step towards a solid programming foundation.

The first thing to understand about HTML is that anyone interested in web development can start learning immediately. There are tons of online articles and guides designed to help people learn about this basic but important front-end coding language. Next, it is important to add complementary coding skills such as JavaScript and CSS, in addition to investing in paid instructional classes.

Although most prospective programmers do not realize it, there is a good chance they have some exposure to HTML. Those who use sites like Myspace, where they could customize their pages inside commands such as '<>', are, in reality, using HTML code.

Essentially, in HTML, any bracketed commands are what programmers call HTML tags, which are fundamental components of this coding language. Whenever developers use HTML tags to create a paragraph or heading, they are creating elements, which are individual HTML components of an HTML webpage or document.

Opening tags and closing tags define these elements, as well as the information, i.e., paragraphs, between them. The HTML elements of a web page consist of different types of media and content, such as:

1. Paragraph: <p>
2. Button: <button>

3. Audio: <audio>
4. Emphasis: <am>
5. Video: <video>
6. Image: <img>
7. Articles: <article>
8. Lists:

To place a particular type of video on the site one is developing; one needs to use its HTML tag, such as, video get <video> tags. For an image, it should be image get <img> tags, and so on. HTML elements can also include the following:

1. Structural elements
2. Meta elements

Developers use the structural elements of HTML to organize a web page's content. Structural elements such as and , describe inline content versus block level content, while <footer> and <header> contain the footer and header content respectively.

Meta elements, on the other hand, are not visible to users when they visit a webpage; however, website developers design them to give web browsers more information about the page, such as date and tie of last modification, document's author, keywords and phrases, and other pertinent information. In addition, Meta HTML elements are helpful when it comes to record-keeping on the programmer's end, in addition to improving the site's SEO results.

Finally, web developers can add attributes to HTML tags to provide more information about elements. A paragraph, for example, which is an HTML element, can have a certain attribute, such as its right, center, or left alignment. Developers include attributes in the opening tag. Attributes include a value in double-quotes and an equal sign, such as <p = align="left">paragraph</p>.

It is important to know all the above turns from HTML code to an actual website. Interestingly, it is a rather simple process. Developers can write their HTML code as a plain HTML document using any word processing program or text editor. Afterward, they save it as an HTML file ending with .html.

They do not need to invest in state-of-the-art computer hardware; rather, and

a decent PC will do. Those saved .html files fill to form the foundation of a site's web pages, which developers launch as a live website through web hosting. Web browsers viewing or visiting these pages will translate the text and tags and process them into what the end-user sees on his/her screen.

However, to achieve better results beyond static web pages, developers need to use other coding techniques and languages. Nevertheless, HTML is a foundational and key aspect of website development.

- **CSS**

CSS also called cascading style sheets is what gives HTML code a visually appealing look that draws in end users. In other words, this front-end coding language determines the presentation of the elements of HTML discussed above on a web page. Essentially, it makes a web page look like a white background with text and hyperlinks.

For example, people who have tried loading certain sites, such as Amazon, on a bad day, often see a white page with a long list of large black texts with nothing else, apart from a few blue links. This is an example of a website loading without styles that bring interesting designs onto the site.

Technological advances brought CSS pre-processor, which is a new trend in website styling. In this case, pre-processors, which include stylists, Sass, and Less, are scripting languages that compile cascading style sheets for the browser. Since they expedite the website development process, they are extremely popular with programmers.

In addition, they allow for some coding logic. CSS is also scalable, which means that it is helpful when it comes to huge websites. Most experienced front-end developers understand the flexibly and power of CSS to suit many different scenarios, such as enterprise software programs like NBCUniversal or Groupon and smaller single-page sites.

From the above, it is evident that CSS defines the style for web pages, including the layout and design, as well as display variations in terms of different screen sizes and devices. It saves these styles in external .css files, which developers can use to change the look of a website.

- **JavaScript**

Due to the awesome power of different tools, libraries, and other things that make software development easier, many coders start developing programs

and applications without having a clear idea of how things work beneath the surface. JavaScript, in particular, tends to encourage this sort of behavior.

JavaScript is a coding language that allows developers to add complex features and capabilities to a website, such as interactivity and dynamic elements. Parsed and constructed information from both HTML and CSS in the source code executes JavaScript, which then triggers any specified variables and events.

In other words, this coding language focuses on contexts, functions, and scopes. As a dynamic scripting language, it often behaves differently from other front-end coding languages, which often causes confusion. For example, it can behave differently when it comes to:

1. Global context
2. Object context
3. Function context
4. Constructor

While it is one of the most popular coding languages out there, it is quite complex. Essentially, its high-level development tools attract many programmers, but, unfortunately, some of them tend to ignore the more tiresome or difficult aspects of this front-end programming language.

Nevertheless, learning about JavaScript is, without a doubt, beneficial to software developers. One of the things that separate successful coders from mediocre coders is the willingness to understand the weird or complex aspects of this programming language. Although advancements in JavaScript happen frequently, its fundamentals remain the same.

In fact, its fundamentals are the foundation of most coding tools. Having a good understanding of these fundamentals gives programmers a wider perception and transforms their view of the software development process. Therefore it is gaining prominence when it comes to website management.

This updates the Document Object Model and renders JavaScript in the web browser. However, both HTML and CSS form the foundation of a website's page structure. Essentially, JavaScript will make the final changes and tweaks.

Interestingly, as the online world continues to evolve and progress, the use of JavaScript continues to experience a remarkable rise. Not too long ago,

websites were static, and programmers only used HTML and CSS coding language to build them. However, nowadays, most websites are interactive, which usually means using JavaScript.

An analyst called John Mueller from Google Webmaster Trends recently said that JavaScript is not going away. Due to the wide use of this coding language, developers should make a point of learning everything about it in a proactive and positive way. In addition, it is not solely for websites or software developers to learn. Rather, it is for businesses and people or who want search engines and prospective consumers to access their site content.

When a web page loads, it creates the Document Object Model, made up of objects and nodes that map out various attributes and elements on a page. Other programs can manipulate and modify this page in terms of styling, content, and structure. Essentially, JavaScript allows for the alteration of the elements of a web page's Document Object Model.

Back-End Programming Languages

When looking for coding jobs, most programmers look for various requirements, such as methodologies, frameworks, and coding languages, most of which may differ. However, there are two components of software development that remain constant for all jobs, and these are back-end and front-end.

The back-end of a program or application is, in reality, the enabler for its front-end, in spite of the fact that front-end technologies hoard most of the attention; at least they have been in the recent past. Back-end coding languages handle most of the program or application logic, database interactions, performance, and much more.

Some of the higher-paying programming jobs require full-stack skills and experience, which simply means a combination of back-end and front-end skills. From a professional point of view, it is important to differentiate between the two.

From the earlier explanation of common front-end technologies, it will be easy to understand the difference after learning about common back-end coding languages. Some of these include:

- **Ruby**

This is an open-source, dynamic back-end coding language, which focuses on

productivity and simplicity. This is why open-source programming languages are so popular. Ruby is easy to write and elegant syntax that has a natural look. In addition to being dynamic, it is a general-purpose, object-oriented, and reflective coding language.

Some of the interesting facts about this high-level programming language include:

1. Yukihiro Matsumoto created it in the mid-90s with the aim of developing a more powerful scripting language than Perl, in addition to being more object-oriented than Python.
2. Most people confuse it with Ruby on Rails; a server-focused language designed to implement online applications, developed by Ruby and licensed by MIT.
3. The name Ruby originated during an online chat session between Yukihiro Matsumoto and Keiju Ishitsuka in February 1993, even before there was a code written in this programming language.
4. Instead of using 'null', Ruby uses 'nil'.
5. Rubygems, its central library management system, manages its packages and libraries.
6. Developers can use Ruby Interpreter as a calculator.
7. It supports various coding paradigms, including functional programming, object-oriented programming, and procedural programming, which makes it a popular and unique coding language.
8. It supports both complex numbers and rational numbers, which gives it the ability to solve complicated mathematical equations.
9. Developers using Ruby can use both Duck typing and dynamic typing.
10. It has in-built support for code embedding.

- **PHP**

This is a server-side coding or scripting language used to develop web applications, dynamic websites, or even static websites. Hypertext Pre-Processor, or PHP, originally meant Personal Home Pages. Only a server with PHP installed can interpret PHP scripts, which means that end-user

computers accessing these scripts only need a web browser.

In addition to having PHP tags, a PHP file ends with a '.php' extension. A script, in terms of back-end coding, is a series of instructions interpreted at runtime. Therefore, developers embed scripts into other software platforms with the aim of performing routine tasks for programs or enhancing program performance.

While client-side applications interpret client-side scripts, the server interprets server-side scripts. For example, JavaScript is a client-side script, and PHP is a server-side script. However, developers can embed both of these coding languages into their HTML pages.

PHP is a developer's poison to the web because of the following reasons:

1. It is a free and open-source coding language
2. In has in-built support for MySQL
3. It boasts a massive community document
4. Compared to most other back-end coding languages, it has a relatively short learning curve
5. By default, most web hosting servers support PHP
6. There are regular PHP updates to help developers keep up with the latest technological advances and trends
7. Developers only need to install it on the server, but end-users do not need to have it installed
8. It is a cross-platform coding language, which means that developers can deploy their programs on different operating systems, such as Mac OS, Windows, Linux, and others

According to some figures, there are more than 20 million applications and sites on the web built using this scripting language.

- **Python**

Python is one of the most beautiful and elegant coding languages out there. Software developers love using it because it seems to express their ingenious thoughts and is somewhat natural. In addition, it is ideal for multiple applications, including machine learning, web development, data science, and more.

Some of the basics or fundamentals that make this coding language so powerful include the ability to determine or take advantage of the following:

1. Variables
2. Conditional statements
3. Iterating or looping

For newcomers into the field of software or application development, choosing the coding language to focus on is perhaps the most important decision. Often, most of them gravitate towards Python or Java. Both coding languages provide unique advantages for end-users and programmers.

However, due to its amazing intuitive programming style, Python is more user-friendly than Java and most other coding languages. In addition, it is the ideal coding language for beginners looking to pursue a career in software development. However, just as two logs produce more heat than a single log, understanding both coding languages will place one a step ahead of most software developers.

This discussion, however, is focusing on newcomers to the field of back-end coding languages. In this case, it is better to focus on Python because it is less complex. Python is a high-level, back-end coding language, which makes a strong case for readable and understandable code.

In addition to supporting functional and imperative coding, it also supports object-oriented programming. It is structured supportive and allows for logic programming and multi-paradigm coding language, as well as so-called, 'magic methods.' Some of the most important features of this back-end programming language include:

1. Optional object-oriented programming
2. Duck typing
3. Easier to read and understand relative to other back-end coding languages
4. Uses white space to designate or point to the beginning and end of each block of code
5. Ability to assign strings to variables that once held integers
6. Programs or applications run faster
7. Ability to compile native bytecodes

8. Creating a program requires less code

However, it is important to note that many platforms do most support this coding language. In addition, it is somewhat slow when it comes to execution.

Best Programming Languages for Specific Projects

With time at a premium and tons of coding languages out there, it is understandable why newcomers in the field of coding find it difficult to choose the right coding language. The first thing they need to understand is that there is no perfect coding language.

Therefore, software developers should pick a coding language that best fits their knowledge and project objectives. Essentially, coding languages are tools for building applications and software for websites, mobile applications, robotics, machine learning, big data analytics, artificial intelligence, and much more.

Therefore, as stated above, it is important to choose the best coding language for each specific project. When it comes to choosing a coding language for a website, for instance, web developers should remember that website development involves two main parts. These are the front-end and back-end.

As discussed earlier, the front-end deals with things that end-users see and interact with on a web page or application, while the back-end deals with behind-the-scene things that deal with functionality, such as logic and data management. Web developers, therefore, should choose a coding language such as JavaScript, which is viable on both back-end and front-end web development.

When it comes to data analytics, artificial intelligence, and machine learning, things get a lot more complicated. However, with the right understanding of different coding languages, it is possible to select the best coding languages for any of these applications.

Currently, the world is in the midst of data, machine learning, and AI revolution. New technologies and advancements in these fields of technology seem to pop up every day. Therefore, it is important to choose the right coding technologies and languages to take advantage of this information and technological revolutions.

Java, for example, is ideal when it comes to big data processing, while

Python is great for machine learning. Scala, on the other hand, is a great coding language for applications meant for data streaming, such as online gaming streaming. However, it is important to understand that these suggestions are not rigid.

For instance, software developers with unique project requirements can use Python in conjunction with Spark to build a faster, user-friendly, and more efficient application or program.

When it comes to mobile apps, it is obvious that two massive ecosystems exist, which are iOS and Android. The current era is a mobile-first era. Although iOS commands a more lucrative market niche, most people in the world use Android. For software developers, the growth of this market is a massive advantage.

Essentially, there is a huge demand to develop software and applications for both platforms. They cannot really go wrong focusing either way. To develop Android apps, Java is the favorite coding language. For iOS, on the other hand, Swift is the current language of choice.

For hardware coding, robotics, and systems programming, coders should go for high-performing coding languages with low-level access to hardware and a small footprint. C and C++, for example, focus on the underlying technology, such as embedded systems, digital signal processing, operating systems, and device drivers, making them ideal for such applications.

In the same way as French is the language of love and English is the international language, different coding languages are suited for different purposes. The field of software development tends to evolve so fast that it can seem a bit overwhelming for people who want to pursue a programming career, especially when trying to keep up or stay ahead of the latest trend. However, this super-fast pace is okay because advances in coding practices and technologies are actually helpful to passionate programmers.

Chapter 5: Fundamentals and Techniques for the Most Common Coding Languages

In computer programming, you will regularly encounter various fundamental concepts and techniques by which you must abide by completing your tasks effectively. It is vital that any aspiring programmer learns and understands what his or her final software product requires before commencing on the coding process.

These fundamentals invariably represent the core principles that are standard across all the various programming languages and platforms. They provide structure to the program, which in turn translates into the eventual application software. Some of the elementary principles to consider will include the following:

1. **Define all the necessary variables**

A variable is suggestive of the aim of your program. Before writing down any lines of code, a programmer must define his or her objectives. Variables are how programmers meet some of their aims. First, you must know what you want from the program before setting off on your own. A variable or more is an essential concept that you must possess, depending on your desired task. This term represents a prospective value that is currently unknown but becomes flexible when the program executes itself after coding. Once an amount attaches to a variable, your program's outcomes or outputs become clearer.

As the programmer, you need to consider every possible task that the end-user might need to get done. The range of possibilities for variables is usually extensive. Whenever the client uses your final product, he or she will expect results based on his or her input queries. Any input that seeks to affect the variables will inevitably result in a specific value output from the variable. Computer programming is principally an exercise in trying to assign values to previously unknown variables based on logic. Therefore, the concept of variables becomes doubly important in the event of such a scenario.

2. **Use conditions to identify choices and select appropriate decisions**

The use of a feedback loop is a critical component of any programming language. During the execution phase of your final product, the software

must handle every kind of scenario that would be in store from the end-user. Most interactions between a program and the client involve making decisions based on a particular set of conditions. The loop represents some form of an iterative cycle that is essential for the program to make sense of any data input.

The articles "if" and "then" are very crucial when writing any form of program code. Its usage sets up a specific condition for which a particular action warrants a sensible output. For instance, the occurrence of one expected effect is dependent upon the satisfaction of individual requirements. A few programmers tend to use other selective codes within their programs. In this case, there is no restriction on the choice of feedback command. Hence, identifying options followed by its correct execution are basic constructs for any lines of program code.

3. Use iterative program codes and loops

Having a series of program codes that you can trust to execute a specific task effectively is essential. The same commands must result in similar outputs from the program during the run phase. Therefore, the use of particular program codes that signal an iterative execution for the same task becomes crucial. Beware of the likelihood of different outputs from the same type of tasks scheduled for the program. For you to achieve proper loops that behave in the right manner, you may need to include specific articles within your program code. Such sections include the use of the word "for" rather than packing your lines of code with "while."

However, some individual programmers tend to use other means of achieving useful loops. In this case, no restrictions apply to such programmers as long as they make the expected purpose of repetition. JavaScript and Python are two of the most popular languages that employ iteration in a majority of cases. Running a script to its complete execution is a process that wholly depends on repetitive loops of program code. Hence, this regular application of iteration is as fundamental as defining variables as well.

4. Make use of arrays and indexes

A good programmer knows when to include arrays in his or her program codes. These arrays represent a set of identical kinds of data or information from which the program picks an item relevant to a given task. The

interaction between your program code and the source of data occurs using indexes.

These indexes form the vital aspect of any program command that seeks data from your storage sources. An example of this concept in action is during random data selection. A programmer randomly assigns a nominal value to data sets of the same kind. Whenever he or she wants to retrieve a specific item from the data set, he or she calls up the nominal number.

The output that results from this process will have particular value depending on your choice. Therefore, in this case, the random numerical figures represent indexes that link directly to specific values contained in a data set. Remember that the data set is full of distinct variables, but of a similar kind.

The distribution of the arrays takes the form of strings within the data set. Beware of specific languages such as Python that use lists instead of arrays. However, a programming language such as JavaScript is excellent in its use of binary indexes within its data arrays. This use of binary options ends up providing the program codes with a higher level of clarity during command executions.

5. Employ Boolean concepts

As a programmer, you need to confirm whether your program codes obey the rules of logic. This flow is essential for the product to perform its expected tasks correctly. During the execution phase, a completed software product will run based on its underlying program code.

Proper Boolean logic when writing down your program codes is, therefore, necessary. The line of program codes must make enough sense for the ultimate product purpose to hold. During this stage, you should make sure to apply your conditioned articles correctly. Such items include "or" as well as "and."

At the end of the coding process, a program usually treats its commands as instructed. These instructions typically resemble binary circumstances. The product's accomplishment of a specific task depends on the conditions allowing the task's execution holding. If not, then the other binary option is an absence of any response from the product. According to the program, this lack of response is the only logical option, as well.

6. Pay more attention to mathematics

Knowledge of math is an essential tool to help you learn to program quickly. There are concepts of mathematics that often apply in the course of programming. Trigonometry is one such mathematical discipline that is common regardless of the type of programming language in use.

As mentioned earlier, maintaining logical sense is a vital aspect of the functionality of any eventual program. Math and its associated disciplines enable the programmer to make an appropriate analysis of his or her program code. This analysis allows for the correction of any illogical arguments within the lines of code.

A good programmer typically has skills in math that are above average. Specific programming languages are purely functional and, therefore, wholly rely on mathematical arguments that always hold. Two famous examples of program languages regarded as strictly practical are Scala and Haskell.

Mathematics invariably becomes a precondition for involving yourself in computer programming. The use of pseudocodes is a routine that is entirely mathematical since it includes some algebraic elements. You need to be excellent at developing these rough drafts of your potential program codes using references from both geometry and algebra.

Once you are ready with your pseudo code, the insertion of specific values and variables during the primary coding process suddenly becomes easy. It also enables you to make corrections to potential errors and mistakes quickly because you can identify them fast. Another reasonably necessary concept within this category is knowledge of modular arithmetic.

This concept allows for a restrictive number of potential outcomes from a single task. These finite probabilities are a result of engaging in successive serial divisions down to much smaller values. Modular arithmetic also enables a programmer to manipulate data values beyond their maximum levels. For instance, you can reverse the angular direction of a circle from three hundred and sixty degrees to zero.

7. Be mindful of object-oriented programs (OOP)

As opposed to functional languages and programs, these programming skills deal with objects. While technical programs maintain their singular mathematical logic, OOP has various character traits to consider. Some of these additional features include processes, field sizes, and manipulative methods of particular data sets.

Other data attributes tend to expose OOP to an increased likelihood of developing bugs during the coding process. Haskell, being functional and logical at the same time, often has minimal coding errors when compared to OOP. A known and widely popular type of OOP is the C version of the programming language.

8. Learn the scaling skills of random integers

When developing applications, program developers often assign random numerical values to integers for specific purposes. This practice is common among software developers who engage in programming, explicitly using the Python language. It has a functionality tool inbuilt within the language that allows for the spontaneous production of random numerical digits.

However, some programming languages do not allow for the automatic generation of these random numbers. Therefore, your skills at scaling integers become crucial in such a scenario. You must learn how to fit data sets and other information on your display using random numbers. These numbers typically perform functions other than ensuring alignment within the screen. As previously mentioned, these random integers might come in handy when indexing your arrays of data strings, as well.

For you to achieve a higher level of reality within your random integers, you need to practice this process in iterative cycles. Over an extended period, the randomness of your integers gradually grows closer to the real-world simulation as possible. Before long, you become masterful at scaling these numerical values.

Beware of the skewed nature of numerical values between zero and one, which results in lower numbers when squared. These lower values alter the expected trend of the range of squared values. Therefore, when dealing with such integers, you need to make the relevant adjustments before proceeding. An appropriate alteration to reverse the trend in the correct direction involves subtraction of those smaller values from one.

9. Interact with text exhaustively

Computer programming involves passing commands and instructions on specific tasks to some given machine hardware. As a result, programming languages differ from human language. A machine understands instructions in the form of program codes, cryptographic content, and other cipher

techniques. It is, therefore, essential that prospective computer programmers also understand how to manipulate text for effective machine communication. You need to possess in-depth knowledge of how to develop these lines of program code.

An encoding standard that warrants profound understanding is the proper command of ASCII. It is an acronym for American Standard Code for Information Interchange. Another cryptographic skill to have is the comprehension of Caesar shift cipher for encryption. These techniques are the means through which programs interact with each other. You must know how to move within the field of machine language seamlessly. Once you master these languages and become adept at coding, your final product ends up running smoothly.

10. Use bitwise logic

This logic involves the incorporation of a series of separate data values and data sets into binary options. This action results in an increased amount of storage space and hardware memory freed up from different locations. The use of bitwise logic also allows for the quick interpretation and execution of commands and program instructions by the application.

When combined with purely functional languages, the level of logic increases exponentially. This increase, in turn, causes a paradoxical yet significant drop in the incidences of errors and bugs within the program code. Due to these advantages, it is highly advisable to apply bitwise logic in conjunction with other logical techniques such as the Boolean system.

Chapter 6: The Best Practices for the Most Common Coding Languages

According to some people, coding is an art, others say it is a science, and still others think it is both. Whatever the case, without practical focus and the steady hands provided by engineers, coders/programmers would only offer artistic visions and scientific theories. Fortunately, thanks to engineering languages people now own devices that can tap into the world's knowledge with a few clicks or taps.

Coding languages have been around for many years. Following the development of the first computer in the 1940s, programmers used hand-tuned computer programs. However, these language programs required a lot of intellect, tie, and effort. The first coding language came in the 1950s.

Over the next few decades, software developers introduced many coding languages, such as COBOL, SQL, FORTRAN, Ada, Wolfram, Eiffel, and C++. However, the Internet age took coding languages into another level, with the widespread of scripting languages, rapid application-development languages, and functional languages.

Scripting languages, after years of implementation and optimization, became the most popular coding languages used on the web. Over the last two decades, Ruby Rails, Java, PHP, and Haskell grew in popularity, thereby laying the foundation for the programming languages used today.

Coding languages continue to evolve, and the most popular languages today include Swift, C#, CSS, and Java. In fact, there is a move to make all coding languages open source and improve functional coding support in common coding languages.

Whichever coding language a programmer chooses to use, he/she will need to follow certain best practices to ensure the finished product is perfect in both function and form. Some of the best practices for the most common coding languages include:

1. Consistent Indentation

Having chosen an indentation style to use, it is important to stick with it throughout the project. There are many established indentation styles; however, the style a programmer picks is not what really matters.

The most important thing is to keep it consistent to produce a code that is easier for developers to navigate through and quickly identify issues or

mistakes. When a programmer joins a project with an established indentation style, for example, following that style is the best practice.

2. Clear Documentation and Commenting

There have been significant improvements in integrated development environments, IDEs, in the last few years, which make it easy to include comments in the code one is creating. Clear comments make it easier for other programmers to identify the purpose of different functions within the coding.

This is very helpful when it comes to group programming efforts and open-source coding efforts to develop software. There are many integrated development environments to choose from, as well as many commenting tools. However, it is important to avoid unnecessarily cluttering the backend of software through obvious comments.

3. Avoid Deep Nesting Methods

Using tabbing to nest code helps to organize lines of code in an easy-to-read manner. Deep nesting, on the other hand, makes it more difficult to read code because it involves using multiple tabs to organize different lines of coding, making it more confusing.

Minimizing the number of tabs in software coding improves readability, which allows software developers to navigate and access certain lines of code ore easily. This is especially helpful in situations where they need to make changes in the code or fix errors.

4. Use Short Lines of Code

Like avoidance of deep nesting and consistent indentation, limiting the length of each line of coding can significantly improve readability. Newspapers, for example, execute line length limitations quite well. It is easier to read narrow but tall columns of text than long, horizontal lines of text. Limiting line length makes the coding language easier to read and understand.

5. Organize Files and Folders

It is possible to code an entire program in one folder if a programmer wanted to; however, it would be extremely difficult to keep up and navigate. The best practice is to use a system or framework of organized folders and files, which

will make the maintenance and management of the program much easier. It will also be helpful for any other programmer who may join the project.

6. Complete Code Refactoring

This refers to cleaning up the software code, and it is important for every programmer to practice code refactoring. However, it is important to understand that this is not a way to add functionality or fix bugs. Rather, it is a way to improve the quality, cleanliness, and readability of recently written code.

The best time to do this is the day after writing the code because it will still be fresh and familiar to the programmer. After refactoring the code, it will still be readable and reusable even months later. In the field of software development, the best practice is refactoring early and refactoring often.

7. Testing

Every software developer knows how it feels to have a great vision of the program architecture in his/her head and spew out lines of code like a processing machine. At such times, programmers can never turn their vision into code fast enough. While this code might be brilliant, it is often unfinished and hairy in places.

In addition, the programmer might not remember places where he/she needs to fill in the gaps later. Therefore, for all of its artistry and brilliance, the program will not be ready to launch. The way to make it shippable is through rigorous and disciplined testing through any of the high-quality application security testing tools.

There are many strong automation features and protocols built by development teams over the past few years. Software developers are employing new integration mechanisms that take their software code and start analyzing it as soon as they check it in. these testing tools ensure that code moves forward, as long as programmers build good unit tests.

If a programmer makes a mistake, the testing automation robot will identify it and alert the programmer. When the code passes all the unit tests, the programmer will be confident that it will not fail, at least not in the anticipated ways. There is no way to ensure that one's code is complete; however, testing will help one catch the obvious mistakes.

8. Use Repositories to fix Mistakes

Programmers often make mistakes and wish they could go back in time to fix them. They follow a certain path, analyze each line of code and add new code, only to find that the original code was much better. Fortunately, by committing the code to repositories during the coding process, it will make it possible to experiment with and improve code without worrying that they might be making a huge mistake.

These version control systems also let programmers synchronize their work, track differences, and merge their code with other developers' code when the time comes. Without this tool, development teams would have a difficult time building reliable code, in addition to being afraid of experimenting with new features.

9. Use Good Development Methodologies

Software developers would not be able to design and build software without a way to merge their brilliant intuitions, instincts, and visions into a carefully planned and rational workflow. There are many opinions about the many different types of development methodologies. Some of these opinions oppose each other.

Some programmers, for example, strongly believe that people cannot build powerful software without taking advantage of the flexibility of agile project management methodologies. Others, on the other hand, think agile methods are too arbitrary and capricious. The agile process has many great aspects; however, when too many developers wander off the path, it can go terribly wrong.

Software developers have been thinking long and hard about the best way to work together as a team to write code. Unfortunately, they still lack a consensus; however, this does not mean that development methodologies lack merit.

Nowadays, business ambitions are so large that organizations need numerous people working together on projects. Without proper coordination, the process of developing software would be chaotic, to say the least. Therefore, software developers can choose any methodology they like, as long as they choose one and stick with it throughout the project.

10. Code Logic Must Live On

Any type of software will have limitations and flaws; fortunately, age is not

one of the. Organic material will rot and metal will rust; however, the logic in software code will never die. Some IBM mainframes, for example, run COBOL code written decades ago by programmers who never lived long enough to use Facebook or Twitter. However, their brilliant code continues to live on.

Software only encounters age-related problems when it does not have new updates and features in the current software product or when it is no longer compatible with modern systems. Fortunately, code maintenance will allow dated software to remain functional and useful; however, this begins with good software engineering.

When software developers write well-documented software programs with modular interfaces, their code will keep running for many years. In a certain way, it is like downloading their soul into the matrix.

Software development, in most cases, is a team effort; therefore, it is important to employ the best coding practices and methodologies for overall success. The coding practices outlined above are a good place to start for programmers looking to improve the power and quality of their code.

In spite of the coding language, one decides to use, these best practices will improve the coding process and quality of one's coding. In addition, the completed project will be easy to understand, navigate, and modify. Consequently, it will lead to the success of the software functionalities, as well as the success of the project team involved in debugging and maintaining the code in the future.

Coding Industry Best Practices

Coding industry best practices provide benefits to the whole organization, project managers, project teams, and programming leads. Understanding the best coding practices that meet industry standards helps software developers understand how to work within their applications, which is important for organizations that want to grow and accelerate product delivery.

These practices also help programmers control complexity, prevent errors, and improve the maintainability of software. There are tons of coding languages and each comes with different capabilities and features. However, the best practices for the coding industry make it possible for programmers to follow good engineering practices in each language.

One of the most important coding industry best practices, for example, is that

software developers have to follow the same commenting conventions, formatting conventions, and naming conventions on all software. These include rules for naming functions, methods, and variables to help them understand how to use a specific variable or function.

Essentially, the coding industry best practices are the informal rules that the programming community uses to improve software quality based on years of learning and experience. As discussed earlier, some software programs remain in use for decades, which might be longer than the original programmers might have imagined.

Therefore, the coding industry's best practices need to help with the initial software development process, as well as subsequent enhancement and maintenance by people other than the original programmers. According to Tom Cargill's Ninety-Ninety rule, the first ninety percent of code accounts for the first ninety percent of the software development time.

His theory aims to explain why any coding projects often run behind schedule. According to him, 10 percent of the software code accounts for the other 90 percent of the project development time. Therefore, anything that can prevent this problem is worth careful consideration.

In addition, the size or scope of a program or project can have a huge effect on developer productivity, error rates, and the level of management required. Some of the best practices for the coding industry include:

1. **Software Quality**

There are many characteristics associated with high-quality software, but some of them can contradict each other; for example, full error checking versus fast development. In addition, different stakeholders may have different or competing priorities. Therefore, different objectives can have a significant effect on both the efficiency and effort required.

According to Weinberg, any software developers often try to achieve certain objectives at the expense of other quality attributes. Some of the main objectives concerned with how well a software program does include:

- Usability
- Maintainability
- Efficiency
- Dependability

In addition, Weinberg identifies several objectives that a good software program should meet. These include:

- It should be adaptable enough to cope with changing requirements
- It should meet its specifications
- It should be efficient enough for its predetermined environment
- Developers should produce it within budget and on schedule

Hoare, on the other hand, identified 17 objectives related to the quality of software. Some of these include:

- Precise, accurate, and clear user documents
- Clear definition of the software's purpose
- Conformity to preset standards
- Simplicity of use
- Minimum development cost
- Kind to errors and difficult to misuse
- Fast enough for its purpose
- Early delivery
- Brevity
- Extensibility
- Reliability

2. Prerequisites

Before starting to code, software developers need to complete all the prerequisites or at least ensure adequate progress to offer a solid foundation for the coding process. Failing to do this will result in an unsatisfactory software program. What happens before the coding stage is of critical importance to the success of the overall project.

Prerequisites cover issues such as:

- The choice of coding language
- The development of life-cycle and structure
- Detailed design
- Overall software architecture

- Software requirements

For a small project involving a single programmer, however, it might be helpful to combine design with architecture and adopt a simple or short life cycle. A methodology for software development is a framework or guideline used by programmers to plan, structure, and control the life cycle of their projects. Some of the common methodologies include:

- Agile software development
- Extreme programming
- Waterfall
- Spiral development
- Iterative and incremental development
- Rapid application development

The first prerequisite programmers need to fulfill before coding, however, is establishing a clear statement of the problem the software needs to address and solve. It is important to aim for an unambiguous, precise, clear, and complete written specification.

However, software developers should understand that this target might change or prove impossible to achieve. This is why Sommerville stresses the difference between detailed software requirements and less detailed requirements, as well as non-functional requirements, such as response tie, and functional requirements, such as update a record.

When it comes to software or program architecture, Hoare suggests that there are two main ways of building a software design. The first way is to make it simple enough to prevent any limitations, and the second way is to make it so complex that no deficiencies exist. Contrary to how it might seem, the first option is more difficult to achieve.

When it comes to software architecture, the aim is to decide when one needs to do and the program component needed to achieve one's objectives. This is especially important if the system has more than one program. Essentially, the architecture will determine the interface between different programs, as well as considerations for any additional interfaces.

Programmers should consider any non-functional software requirements, such as maintainability, reliability, and response time, at this stage. Other

stakeholders interested in software architecture include end-users and sponsors. It gives them an opportunity to ensure developers are meeting their system requirements.

The primary aim of the design prerequisite is to fill in details that may be missing in the system architecture. This design should be detailed enough to guide the coding process, in addition to having details of the algorithm developers will use. At the architectural stage, for example, developers may identify data that needs sorting. As such, they need to determine the appropriate sorting algorithm to use.

When it comes to the programming language to use, software developers should understand that there is no perfect coding language. Rather, there are coding languages best suited for certain purposes. Therefore, it is important to understand the problem at hand and the associated programming requirements to find the coding language well suited for the solution.

Essentially, the process of choosing the right coding language starts with identifying the actual problem and determining the requirements and their relative importance. This is because it might be impossible to satisfy all requirements equally well. Afterward, one should measure available coding languages against the list of system requirements and choose the most suitable language.

That said, several different coding languages might be suitable for different areas of the project. In this case, if the coding languages permit, it may be helpful to combine them within the same project. Essentially, every coding language has its own limitations and strengths; therefore, it is important to identify them before choosing a language to use.

3. Coding Standards

Establishing coding standards or conventions before starting to code is another coding industry best practice. It is almost impossible to change or modify code to meet coding standards later. Each coding language has its own coding standards; therefore, it is counterproductive to use the same standards across different coding languages.

The application of these standards is particularly important for projects involving several programmers. It is easier for a software developer to read code written by another developer if it follows the same coding standards.

4. Commenting

Due to enthusiastic programmers or time restrictions that demand immediate results, system developers often forget or ignore commenting. However, developers working as a team often find it helpful to add comments since software programming usually follows stages or cycles, and several people might be working on the same module.

5. **Naming Conventions**

It is important to use the right naming conventions. Often, software developers use X1 or Y1 to name their variables and fail to replace these symbols with meaningful ones, which tends to cause confusion. Using descriptive names is a good coding practice.

6. **Keep it Simple**

Writing simple code is a coding industry best practice. It does not make sense to write complicated code to achieve a simple objective because another programmer might modify it in the future. In addition, a complicated logic might not make sense to a different programmer; therefore, it is important to keep it as simple as possible.

7. **Portability**

Software developers should avoid using literal or hard-coded values referring to environmental parameters. These include absolute hostnames, file paths, UDP or TCP ports, URLs, file names, and IP addresses. If they do this, the program will be unable to run on a host with a different design. Careful software developers configure such parameters for hosting environments outside of the system proper.

For most organizations, good coding practices include using the same coding languages throughout the development process. The advantage of doing this is that they can monitor and tailor these practices for their specific set of coding languages. This makes the process of identifying and fixing errors simpler, faster, and easier.

Chapter 7: How to Compile Codes

To reduce the complexity of designing and building software programs, compilers translate code written in a language suitable for human software developers into low-level machine language. Interestingly, programs written in machine language tend to be significantly longer than equivalent programs written in a high-level language.

In other words, compiling is the process of taking written source code from a compiled coding language and translating it into an executable program. This allows computer systems to run and understand the program without the need for the coding language or software used to create it. Compilers compile programs for a specific platform but not on other platforms.

For example, a program compiled for an IBM platform will only work with computers that are compatible with IBM. The same thing applies to the Apple platform. Grace Hopper developed the first compiler while working on the Harvard Mark 1 computer.

Another benefit of using a code compiler to interpret high-level language is that it can compile the same program to run on many different types of machines or into many different machine languages.

However, programs written using any type of coding language and automatically translated by a compiler to machine language may run a bit slower than hand-coded programs written in machine language. Therefore, some system developers still write some time-critical programs partly in machine language.

What is a Compiler?

This program translates or processes human-written code into machine-executable code or language that a computer's processor uses. The written code, however, needs to comply with the syntax rules of the coding language used to create it for the compiler to work successfully. This is because the compiler cannot fix code with errors or bugs. If the code has a mistake or incorrect syntax, the compiler simply will not compile.

Usually, software developers write code in a language such as C++, C#, C, or Pascal one line at a time. The file created, therefore, contains source statements. Software developers then run or execute the appropriate compiler and choose the file that contains the source statements.

A compiler's complexity, however, depends on the syntax of the coding

language, as well as the level of abstraction provided by that coding language. A compiler for C# or C++, for example, is more complex than a C compiler. When running, the compiler first analyzes the syntax of each language statement and then builds the output code in one or more successive stages.

In the process, it checks to make sure that source statements referring to other statements have the right reference in the final code. Traditionally, programmers called the output of the compilation object module or object code. In this case, the word object has nothing to do with object-oriented programming; rather, the object code or module is machine code the processor uses to execute one instruction at a time.

The Java coding language introduced the possibility of compiling bytecode, which is an output code that can execute on any computer platform with a bytecode interpreter or Java virtual machine. This allows it to convert the output code into instructions that the actual hardware processor can execute. This virtual machine or interpreter can optionally recompile the bytecode using a just-in-time compiler at the execution platform.

Some older operating systems required an extra process after compilation. This involved resolving the relative location of data and instructions when running several modules at the same time and they cross-reference to each other's data or sequences. Programmers called this process linkage editing, and its output was the load module.

Compilers work with what the coding community calls higher-level languages and 3GL. An assembler, on the other hand, works on programs written using the assembler language of the processor. As stated above, a compiler is a software program that translates and converts written programming code into binary code for a specific CPU to interpret and execute.

In other words, compilation, in relation to software development, is the process of transforming programming code into machine code. When this transformation happens before the code reaches the platform meant to run it, developers refer to it as an ahead-of-time compilation. Before C# and Java, all programming code was either interpreted or compiled.

Interpreted code runs program instructions without compiling or converting it into a machine or binary language. Interpreted code parses the written code directly and uses a virtual machine to convert the source code for the CPU at

the time of execution or uses precompiled code. An example of interpreted code is JavaScript.

Interpreted code, however, tends to run slower than compiled code because it needs to do more work at the time the action is taking place. Compiled code, on the other hand, does not need to do anything more. C# and Java use just-in-time compilers, which are a combination of interpreters and ahead-of-time compilers.

When programmers write a Java program, the just-in-time compiler converts the code into bytecode, instead of turning it into code that has program instructions for a certain platform's CPU. Essentially, bytecode is not dependent on a particular platform; therefore, it can run on any hardware platform that supports Java.

In the same way, C# uses a just-in-time compiler that is part of the CLR, or common language runtime, which controls the running of all .NET software applications. There is a just-in-time compiler in each target hardware platform; therefore, the program will run if the platform can understand the intermediate bytecode language conversion.

- **Pros and Cons of JIT and AOT Compilation**

Ahead-of-time compilation provides faster startup time, especially when most of the code runs at startup. This type of compilation, however, requires more disk space and memory. Just-in-time compilation, on the other hand, targets execution platforms with lower capabilities.

In addition, a just-in-time compilation profiles the execution platform as it runs and recompiles instantly to deliver better performance. It also generates better code since it targets the intended platform. However, it usually executes slower than AOT compiled code.

Processors run object code, which shows when binary low or high signals are necessary for the arithmetic logic unit of the CPU. De-compilers interpret binary language into high-level natural language, while cross-compilers produce object code intended to execute on a system. A language translator, on the other hand, is a type of compiler that translates one coding language into another.

Generally, compilers run four main processes:

1. Scanning

2. Lexical analysis
3. Syntactic analysis
4. Semantic analysis

There are good reasons for keeping these phases separate. Some of these reasons include:

1. Improved efficiency: A lexical analyzer, for example, might work faster than the parser when it comes to the less complex aspects of code compilation.
2. Modularity: There is no need to clutter the syntactical aspects of the coding language with small lexical details such as comments and white space.
3. Tradition: Languages often contain separate syntactical and lexical phases, and the best practice is to separate such elements of language.

The scanner analyzes one character of code at a time and keeps track of where the character appears. During the lexical analysis process, the compiler converts the series of source code characters into sequences of characters called tokens, associated by a particular rule. The lexical analyzer then uses a symbol table to store the words in the code that relate to the tokens.

In the traditional sense, the word lexical means 'pertaining to words'. In the coding world, words are objects like keywords, numbers, variable names, and other programming objects. The main aim of lexical analysis is to make things easier for the syntax analysis stage.

During the syntactic analysis stage, the compiler performs an analysis involving preprocessing to determine whether the tokens generated are in the right order, according to their usage. The syntax is the right order of a group of keywords that can yield the desired result. Essentially, it checks the program code to determine whether it has syntactic accuracy.

Semantic analysis, on the other hand, involves several intermediate steps. These involve the following:

1. checking the structure of the tokens and their order in relation to the grammar in a particular language
2. Interpreting the meaning of the token structure to generate the

object code

3. Parsing and interpreting the entire code to determine whether there is a need for any optimizations
4. Inserting the appropriately modified tokens in the object code to generate the final code
5. Saving the final code inside a file

Reasons to learn about compilers

Few programmers ever need to write or build a compiler for a general-purpose coding language such as SL, C, or Pascal. Therefore, one might wonder why most institutions offering computer science courses often make compiler courses mandatory. Some of the reasons to learn about these amazing tools include:

1. Experts in the field of software development believe it is a topic every computer science graduate should understand to be well cultured in computer science
2. Software developers are builders of programs, and every builder should know his/her tools. Compilers are one of the most important tools for programmers
3. The methods used to build a compiler are also useful for other purposes
4. Sometimes, programmers may need to write an interpreter or compiler for a domestic-specific coding language.

How Compiling Code Works

When software developers run a program, they know that somewhere, somehow compilation of the code is taking place. They might see some outputs during the process, but for the most part, compilation takes place behind the scenes. In a certain way, a compiler is like a little robot all programmers accept and not many of them question its importance.

However, there are those who would like to learn more about the workings of that robot. Compilers take a few different steps to compile code correctly. First, as discussed above, it takes source code and converts it into a series of tokens, each containing letters, numbers, and symbols that make sense to the compiler.

Afterward, it will parse the sequence of tokens to ensure the code meets all the rules of the coding language. In other words, it checks to make sure everything is just right. If any syntax errors exist, this is the point where the compiler will find them.

Having done this, the compiler will build a syntax tree, which is simply a representation of the code's semantic meaning. Using this syntax tree, it will generate output code, which is the translated code in the form of machine language or any other language.

- **Tokenization**

As indicated above, before a processor can decipher a program code, it must divide each symbol and assign it its own token, which is a character or group of characters inside a source file. Essentially, this is tokenization, a process performed by a tokenizer. The tokenizer splits up the original source code and keeps it inside the computer memory.

Given only the tokens, a different programmer should be able to rewrite the original source code, at least to a certain degree. However, tokens need to retain the same type, and if a certain type of token has several values, it should contain the corresponding value.

By themselves, tokens are not enough to perform a compilation. The code compiler also needs to know the context or each token and what different tokens will mean when combined. This is where the parsing stage comes in.

- **Parsing**

This phase of the compiling process involves taking tokens and producing an *abstract syntax tree*, as stated above. The AST refers to the structure of the program. The code compiler can use it to determine how to execute the program in an abstract way. It is similar to the way kids learn to build grammar trees in school.

Actually, this tends to be the most complex phase of the compiling process. A parser is significantly complicated and needs a lot of understanding and hard work. Fortunately, there are programs with the ability to generate a parser depending on how a programmer describes it to the parser creator tool.

Parser generators, however, are specific to coding languages because they generate a parser in the chosen language's code. For example, Bison is a favored parser generation tool for C and C++ coding languages. In short,

parsers try to match tokens to defined token patterns.

The abstract syntax tree determines the operations to perform and the high-level order of code execution, which makes it quite easy to start generating assembly code.

- **Code Generation**

Since the abstract syntax tree consists of branches that almost define the whole system or program, developers can step through those branches and generate the equivalent assembly sequentially. Essentially, they can walk the tree to generate code. When it comes to complex compilers, however, it is not good to walk the tree immediately after parsing.

This is because complex compilers go through more stages before creating a final abstract syntax tree, such as optimization and other phases. Assuming the successful completion of the syntactical and lexical analyses, the remaining phase is generating machine code, which is a complex process.

Compiled executable code should be as fast as possible; however, the speed can vary significantly according to the amount of optimization requested and the quality of generated code. Most code compilers let programmers specify the level of optimization, for example, full optimization for the generated code and quick debugging compiles.

- **Queues and Caches**

Most computer CPUs use a pre-fetch queue where they give instructions to the cache before executing or running them. However, the CPU has to reload the pre-fetch queue if a conditional branch happens. To minimize this occurrence, the CPU needs to generate the code. Therefore, most modern CPUs have separate components for:

1. Fractional number arithmetic
2. Whole number arithmetic

These two operations often run simultaneously to boost speed. In addition to using internal caches, many CPUs also speed up processing through instruction pipelining. If the internal caches can hold all the program instructions within a code loop, then that loop will execute faster than when the processor fetches instructions from the RAM.

- **How Compilers Work**

A compiler's purpose is to translate a programmer's code into a different type of code; however, it has several different layers. There are multi-pass and single-pass compliers. In the early days of computing and programming, computing resources limited the ability to compile. Imagine attempting to compile program code in a PC that only has a hard disk capacity of 1GB.

This is the reason for the creation of single-pass compilers. Since older computers did not have adequate resources, software engineers created languages like Pascal to compile code in a single pass. However, since modern computers have more capabilities and resources, software developers use multi-pass compilers.

Multi-pass compliers consist of three main components:

1. Front end
2. Middle end
3. Back end

However, these are not like the front end and back end in website design and development. In software programming, the front end is where verification of the source code takes place. Here, the source code goes through a transformation to become something the processor can understand.

The middle end, on the other hand, is a bit more interesting. Code optimization takes place here. It takes the output from the front end and optimizes it to make it faster. In addition, it might get rid of useless code or perform some loop transformation. Essentially, it takes the front end's translation and makes it better for the target platform.

The optimized code then moves to the back end, which completes the code translation into machine language. This is how the system determines how to apply its resources to run the code in the most efficient manner. At this point, the compiler stops working and the program takes over.

- **Reasons to Compile Code**

Some software developers make the mistake of trying to learn assembly language. Although a few programmers are skilled at it, for most people, it feels like this coding language is frustrating enough to drive them crazy. Trying to write a simple if-else statement in assembly language is enough to make one cry.

Fortunately, compilers came to save programmers from insanity. They are the reason software developers are able to use many different coding languages without having to stress themselves about machine code. They are also the reason programmers are able to build complex systems and websites without dealing with zeros and ones.

Compilers can take any coding language and transform it into what the CPU understands. Programmers should take a moment to look at the output of the compiled code to understand how much work compilers do for them in the background. They are amazing and powerful software development tools.

- **Difference between Runtime and Compile Time**

Both compile-time and runtime are software development terms referring to different stages of the development process. The stage where the compiler converts the source code into executable code is the compile-time, while the runtime is the stage where the executable is running. Programmers also use these two terms to refer to different types of errors.

Compile-time error checking happens during the compile time, and such errors usually happen due to typing errors. If a programmer fails to follow the right semantics and syntax of the coding language he/she is using, the compiler will identify such errors and alert the programmer by not executing a single line of code until he/she removes the errors or debugs the errors.

Some of the most common compile errors include:

1. Syntax errors
2. Type-checking errors
3. Compiler crashes

Run time error checking, on the other hand, occurs during the run time stage of the program. The program generates these errors when it is in running state. Run time errors cause the program to behave in unexpected ways. Sometimes, they will even kill the program. Another name for run time errors is exceptions. Some of the common exceptions include:

1. Running out of memory
2. Dereferencing a null pointer
3. Division by zero

Some coding languages, such as Pascal, require the compiler to read the source code only once before generating the machine code, while others require at least two passes. Sometimes, the need to make two passes is due to forward declarations of classes or functions.

In C++, for example, programmers can declare a class, but they will need to define it later. Therefore, the compiler must reread the source code to determine how much memory to assign the class before generating the right machine code.

Most modern coding languages, however, have their own compiler or other tools that have the ability to compile the program. Depending on the program's size, it should take a few minutes or seconds to compile and create an executable file, as long as no errors came up in the process of compiling.

Chapter 8: Tips for Learning How to Code

Learn Programming within the Shortest Time

Do you want to venture into coding, but you have more questions than answers on how to start it? You may be worried about what you need to learn and how to identify bugs as well as fix them. Starting may seem to be a daunting task, but with determination and strong will, you will be able to learn. The good news is, there are plenty of resources online that can help you to master coding. The following are some of our best tips that can set you off on the right foot.

1. **Your reasons for learning to code**

Before starting, you need to ask yourself why you want to learn to code. Assess the real reasons as to why you want to venture into coding. Do you want to make a career change? Do you want to develop apps? Do you want to start a company for building websites or a tech startup? The answers to these questions will determine which programming languages you need to master and the amount of time you need to master the language.

Think about your end goal so that you do not find yourself wandering. If you would like to create system software, then you will need to learn C++ along with data structures and algorithms. If you want to shift your career to tech-related fields that require knowledge of coding, then you can attend short-term coding boot camps. You can get useful coding information from interactive tutorials or online courses that you could access free or pay a few dollars.

2. **Choose the right programming language**

After establishing your end goal for learning how to code, it will be easier for you to know which programming language you will go for. Although all programming languages are good, some are more user-friendly and easier to learn like HTML, JavaScript, and CSS. These are good if you want to learn how to develop basic websites.

If you want to generate websites that incorporate payment systems and databases, you should consider learning SQL, JavaScript, PHP, and Python. Learning Java can help you in creating android apps. If you want to be flexible and fit in different fields, you can consider learning JavaScript and Python. The good news is that once you learn the basics of programming you

can learn any language of your choice, so start with one and learning the rest will be easier for you.

3. Pick a plan on how you will learn

You can decide to take online courses like Udacity or attend coding boot camp classes. You can also learn by yourself using the various tutorials available online. However, the problem with online tutorials is that they are too many that you may get confused in deciding which one is the best. Just pick one and stick to it instead of jumping from one to another. You can take advantage of the free online courses to learn the basics of programming before moving to paid courses for advanced knowledge.

You can also acquire a book that can take you through from the basics to real coding. There are different programming e-books that you can find on the Internet. Most of these books offer good guide practices from project design to debugging code. Interactive tutorials are also good because they give you examples in action and this makes it easier to understand as a self-taught person. They simplify the coding concepts and give you relevant exercises to tackle before going to another topic.

4. Learn by hands-on coding do not just read

The best way to learn to code is by doing it practically. Once you have learned the basics, you should try to use the knowledge to build something. You can start by tackling the exercises that come with tutorials or online courses. Therefore, build a project as you continue learning. No matter how much you learn, your coding skills will show in your project.

Use the knowledge you gain or you risk losing it. For example, if you want to develop a website, you can practice using HTML and CSS. Create the basic HTML codes and run the program. What happens? Modify the codes and see what happens again. Remember practice makes perfect and there is no shortcut in coding. You can start small but think bigger.

Although you may not be the best in the first few weeks, you need to be proud of the small progress that you make and be patient with yourself. Strive to write few code lines that are error-free and logically correct. This is a great achievement for someone new in programming. If you are stuck at some errors, you can Google your errors by copying and pasting the error message with quotation marks on Google search box.

5. **Do not ignore the basics**

Remember that you cannot run before you learn how to walk! It is particularly important to master the basics of any programming language because the advanced concepts start building from the basics that you have learned. Therefore, you need to put much focus on the basics so that you are not stuck when you reach the back-end programming.

6. **Code by hand since it sharpens your proficiency**

You should learn to write code on paper then dry run it before transferring it into the computer. Since you would not be able to check if the syntax is correct, you will be concentrating on what you are writing. This method will sharpen your skills in becoming a sound developer. Most job interviews will ask you to write the code on paper.

7. **Get a Mentor**

Another great way to learn to code is by getting the guidance of an expert. You can ask for help from a person who is good at programming and this will make your learning easier and faster. He/she can help you with code feedback or advice. You can find a mentor online or a local coding meets up.

8. **Learn incrementally**

You should not try to learn several languages at once, start with the basic ones like HTML then you move gradually to complex ones like PHP, Python or C#. Stay committed, disciplined and focused. After learning something, you can involve someone to look at your project and accept the feedback and improve.

Web Development Projects a Learner Should Begin With

The greatest way to learn and understand web development is by designing projects. Going through tutorials is not enough if you are not going to implement what you read in them. Therefore, read the tutorials and let those ideas come into reality.

Good projects are those that help you in solving your problems. If you wish to work on projects that will greatly inspire you, we have a list of good projects that you can start with.

1. **The quiz app**

A beginner can do this good project. The app works by requesting you to answer the questions in the app to determine your knowledge on the topic. You can also create this app to test the coding skills of other developers.

User interface

- The user can start by clicking on a button displayed "Take Quiz" or "Start".
- The user then sees a display of multiple-choice questions and the user is supposed to choose one or more answers from the choices given.
- After the user selects the required choices, he/she clicks a button displayed as "Next" until he/she reaches the last page.
- Finally, the user lands on a page that displays the results he/she has gotten from the quiz (passed or failed).

This app can allow the user to create an account where he/she can save the scores. It can also allow the user to add more quizzes to the app.

2. **Calculator app**

A calculator is a powerful tool since it makes calculations easier for humans. You can use JavaScript to create a calculator app that can solve simple arithmetic calculations like addition, multiplication and division using integers.

User Interface

- The user interacts with a page that shows the current number the user has entered or results from the last calculations.
- The interactive page displays an entry pad that has buttons consisting of numbers 0-9 and operation signs "+", "-", "/", "*", "=", and a "C" button for clear.
- The user can enter numbers by clicking on the numbers and then click on the arithmetic operator to display the result of the operation.
- The user can click on the "C" button to delete the last number or the last operation.

3. Christmas Lights

You need to be creative enough to come up with this light display app. You will draw seven-colored circles in a row then you will use a timer to change the intensity of every circle. It works by brightening a colored circle then the next color circle on line goes back to its normal intensity. This will form a causal sequence of colored lights like the Christmas lights.

User interface

- The user clicks on a button displayed as "START" to start the display or "STOP" to end the display.
- The user can alter the time interval as well as the color intensity and he/she can choose the colors to fill every circle
- The app allows the user to customize the size of the circles, and the number of rows to display.

4. Conversion of Roman Numbers to Decimal Numbers

This app will help you to convert the roman numbers to decimal. You can start with seven symbols and assign them some fixed integer values. For example:

- I can represent number 1 then;
- V = 5
- X=10
- L=50
- C=100
- D=500
- M=1000

The User Interface

- The user is required to enter a roman number into the input field and clicks on the "convert button".
- The result of the roman number converted to decimal number displays on an output field and the user can see it.
- If the user enters a wrong symbol, then the app displays an error

message

- The user can convert from decimal to roman and vice-versa

5. To-Do List app

This app can allow you to write down all the things that you want to do and your end goals.

The User Interface

- The user can see a display with an input field where he/she can type the "to-do" item.
- After typing the "to-do" item, the user can click the enter button or submit button then the item is added onto the "to-do" list where the user can see.
- When the user accomplishes a task, he/she should tick against the "to-do" list to show that the task is complete.
- The user can also delete an item by pressing on delete or remove button.
- The user can make changes on the "to-do" list.
- The user can see all the completed tasks as well as those that are active.
- The date of creation of the "to-do" list should also be visible.
- The app saves all the "to-do" items and the updates and it allows the user to retrieve them.

6. Create a Library management system website

This website will automate all the daily work of the library and the website has two sections, the admin's section, and the students/teachers section. Anyone who intends to use the website should register first.

The Teachers/Students User Interface

- The user can log in with their details to access their accounts and personal information which can they can update.
- The user can search for a book by using the author's name or department.

- After finding the needed book, the user can order the book.
- The admin can approve the issue of the book and he/she can see the return date of the book.
- An automatic timer will start on the user's account showing the time left to return the book.
- If the user feels like extending the time of return, he/she can write a message to the admin stating his/her intentions.

The Admins' User Interface

- The admin can see the students and teachers who are online.
- The admin can access the information of the library users.
- He/she an update the book records by adding some and deleting others.
- He/she can see those users whom their date of return is due and sends messages to them.
- The admin can see all the books that are out of the library and the borrowers' details.

The Future of Coding

We all agree that the technology world is moving very fast! 5 years from now the tech world will be different and more advanced. Programming keeps on changing and languages that are more sophisticated are coming so a programmer needs to be on his/her toes to keep up with the market trends. The following are some of our predictions on the future of programming.

1. **It will be more of cloud-native**

A good example is the Google search engine that has thousands of their servers in the cloud. Whenever you search for something on Google, it has to contact its several servers to ensure that you get the correct answer to what you are looking for. Cloud computing is destined to be bigger shortly since it is becoming popular.

Businesses are resorting to having their servers in the cloud because they are fast in accessing information and you will not need to buy, maintain or upgrade any physical servers.

2. **IoT security will get scarier**

IoT technology has grown over the past few years and its future is very promising since it is becoming very popular with the latest devices. However, since most devices are used to access information from the cloud, there is a great risk of malicious people hacking into other people's systems to access their information.

Programmers need to install tough security measures on these IoT devices to ensure the security of information in the cloud. They need to come up with some codes and algorithms in IoT devices to detect and prevent any suspicious activity from accessing information. This will ensure that the whole system is safe and the invention of IoT devices will continue to grow.

3. Video content will flood the web

The HTML standards committee embedded video tags into HTML codes to solve the defects of plugins. Video tags respond very well to JavaScript commands and this makes them easy to program. In the past video content was fixed and you could only watch and do nothing.

Nowadays, web designers have come up with ways that can allow you to interact with the video and control how the story flows through a seamless canvas design. With the invention of machine learning, you can experience high-quality streaming of videos without buffering.

4. Consoles will replace the PCs

Gaming consoles technology has advanced with great video cards and stable software platforms and people now prefer gaming consoles to PCs. Most of the latest household gadgets like toasters are starting to have digital memory. This will help them to remember our preferences. We are predicting a time whereby these household devices will be able to communicate with each other and share information.

5. Data will still be king

Data is and will always be very important. It is what determines the network or internet. What we see and read every day from the internet is from data. Data collection, curating and parsing will remain to be very important even in the future. Numbers are important for decision-making but programmers need to come up with data that is easier to understand.

Because data is important, it is important to put security measures to protect

it. You can do this by adding SSL to your websites and other data encryption methods.

6. Machine learning will dominate

Machine learning is slowly getting into the tech world. Some big businesses are already using it in heaping their big data projects with the support of R programming language and python. For machine learning to work, it requires the presence of programming and codes but the future machine learning may minimize programming.

7. User interface design will get more popular

PCs are fading away day by day and smart devices are replacing them. Soon you will no longer see keyboards and mouse. Room consoles and smart devices like phones and tablets are taking over and people are moving from click to touch. This is because touch technology is faster and it can recognize face and fingerprint which is the new exciting form of security.

Voice control has also heightened and big businesses are using it to interact with their customers. For you to design a powerful user interface, you need great programming skills. User interface designing keeps on advancing with time and now we are seeing the UI with overlapping effect, functional animations and contrasting fonts.

8. Autonomous transportation

We have recently seen cars that are in driverless mode. This is possible with advance programming and we could see other transportations like aircraft using autonomous algorithms to achieve a pilotless mode of transportation. The creation of an autonomous skateboard that is a lightweight electric vehicle is underway. This autonomous transportation will require great programming codes so that they give the intended results.

Security will also be paramount when this autonomous transportation takes off. This is because hackers can hack their systems and divert them to go elsewhere. Therefore, programmers will need to put in place algorithms that can detect any suspicious activity on the devices.

9. The law will redefine new limits

Although there is an excitement in technological advancement, there will be a

challenge in restraining malicious content on the digital platform and even in the real world. Cyber-attacks and identity theft are still on the rise and the security personnel needs to incorporate the work of programmers in fighting these malicious people. They need to think about privacy and the law when writing codes.

10. Stiff competitions between open source and closed source

Although open source is becoming popular among the programmers, closed source is still valued because of its security features. Most of the smart devices running on android use open source. However, a closed source is becoming very popular too because of its support quality and the ability to customize it as well as its security features.

Conclusion

Thank you for making it through to the end of *Python Programming for Beginners.* Hope it was informative and able to provide you with all the tools you need to achieve your goals whatever they may be.

Now you know that a Programming language is a high-level and unique language that a programmer utilizes to instruct computers to perform specific tasks. He or she creates scripts or software programs and uses them to command the computer to carry out particular actions.

Low-Level Languages are basic computer instructions, which also go by the name of machine codes. A computer is not able to understand any form of instruction given by a user in human language; however, it can easily understand codes, which after processing the codes, can give the proper responses to the users.

A high-level language is the kind of language that facilitates the growth of a program that is friendly to the user in a complex programming background. These languages are close to the human language and enable the programmer to focus on the issue that needs a solution. This kind of language mainly focuses on the programming logic instead of underlying hardware elements like register utilization or memory.

A woman named Ada Lovelace established the first computer program in 1883. She developed an algorithm for running a particular ancient analytical engine that was the property of Charles Babbage used to calculate the Bernoulli's numbers. The algorithm enabled the success of this calculation accurately.

The easiest way to identify the best programming languages to learn at any given period is by looking at where the trend is going or listening to what the market is saying. Fortunately, as newcomers begin their journey as programmers, they will start to discover which coding language will most suit their objectives. However, before choosing the programming language to use, programmers need to consider several factors, such as:

1. The projects they are working on
2. Their background in logic and mathematics that might help the learn
3. Whether they want to work for an established company, go freelance, or work with a startup

4. If they are interested in website development, they need to determine whether they prefer working on the back end or front end
5. Whether they want to learn higher-end coding languages that are more flexible and have certain abstract programming concepts or focus on lower-end coding languages that have less abstraction and are closer to the hardware

Some of the most common front-end coding languages include JavaScript, CSS, and HTML. JQuery is another front-end programming language, but it is going out of style. Prospective software developers should not be surprised to learn about legacy projects in their curriculum. This is because they still use the JavaScript library. They will also learn a lot about responsive design, color theory, grid system, and typography.

Therefore, software developers should pick a coding language that best fits their knowledge and project objectives. Essentially, coding languages are tools for building applications and software for websites, mobile applications, robotics, machine learning, big data analytics, artificial intelligence, and much more.

To reduce the complexity of designing and building software programs, compilers translate code written in a language suitable for human software developers into low-level machine language. Interestingly, programs written in machine language tend to be significantly longer than equivalent programs written in a high-level language.

The next step is to begin learning a computer language, if you have not already, and start developing codes or even applications that could end up making you very rich.