I wrote the following for a friend in 2008, so some of this info is out of date now.

I've been thinking about how best to introduce you to the world of web development. I'll describe three main things--the path I've taken to learn web development, the general structure of software, and practical recommendations for where to start.

1. My path

The general path I've followed in learning web development is below. All items but the ones marked with an X remain essential for my work today, so could be useful for you to learn. Of course there are many other unmentioned ways to go about this, and it's good to use the latest books & online materials to your advantage. You can Google for the keywords below for more info.

1999-2000
Study WebMonkey's Information Architecture Plan
Create summary of web design
Study how to write a business plan, business finances, freelancing
Create web pages using WYSIWYG page editors & editing HTML/CSS/JavaScript by hand
Buy & use the following (now dated) books:
* O'Reilly's Web Design in a Nutshell (overview reference)
* Wrox's Professional PHP Programming (tutorial)
* Sams' Pure JavaScript (exhaustive reference)
Install Xampp (Apache web server, MySQL database, PHP server-side language)
Create pages using PHP & MySQL
Create & maintain sites for various clients - continued through 2004

2004
Learn Python
Learn (centralized) version control - Concurrent Versioning System (CVS), then Subversion

(SVN)
Use TikiWiki for my site (a content management system - CMS) - X

2005-2006
Use Joomla (a CMS) for my site & others
Write Joomla components in PHP - X
Write custom AJAX/DHTML

2007-2008
Use (lightweight) "rapid web application frameworks" TurboGears (in Python) & CakePHP (in PHP) - both are patterned after Ruby On Rails (in Ruby), and use the latest programming methods, which are the "best practices."
Use AJAX/DHTML JavaScript toolkits - Scriptaculous, MooTools, jQuery, MochiKit, ExtJS
Learn distributed version control - Bazaar
Pick up lightweight project management tools - Trac/SVN

The path I've taken follows the advances made in web technology over the last 10-15 years (static pages to dynamic database-backed sites to CMSes & rapid web application frameworks), the general hierarchy you might follow to learn web technologies (from basic to more advanced), and the last things in the list are of special value if you need to know (or use) the latest and best technology.

2. The general structure of software - MVC

It seems to me the best way to introduce you to how a website works is to conceive of the website as if it is a software program or application, which in a sense it is, and group the necessary technologies according to how they function within the program. One of the best basic paradigms for understanding and organizing a program today is the "Model, View, Controller" paradigm, or MVC. The Model is where you store your data -- the database. The View is the part of the program that presents the data visually to you, the user. The Controller is where you put your programming logic ("business logic") that decides what data to get from the Model (the database) and present to the user through the View, or what to do with information the user sends into the program through the View (often the Controller will write user-submitted data to the database.) The technologies centrally involved in a website follow this pattern. To give you a little background, first I'll list some of the things you could put into a website and distinguish which things belong more to the server and which belong more to your particular site:

Server
Web server hardware - rent space on a host like WebFaction.com for $0-$10/month
Web server software - serves the site's pages to your site's visitors - on Linux servers it's normally Apache; on Windows it's IIS (Internet Information Services)
Database software - MySQL, Postgres, MS SQL Server
Server-side programming language - Perl, PHP, ASP, ASP.net, Java, Python, Ruby, etc.
Domain name - publicly-accessible address for your site's server - register for $20/year with a registrar like GoDaddy.com

Website
Static HTML pages - stored on server, which serves them to site visitors
CSS - (Cascading Style Sheets) - defines the visual layout of HTML elements (dimensions, color, font, etc.)
PHP pages - server-side programming language dynamically generates HTML pages in response to browser's requests, pages are then served by the server
JavaScript - client side code (runs in the web browser) to enrich the user interface - AJAX & DHTML are popular uses
Flash - another client-side language to provide a rich user interface
Content - text, images, audio recordings; stored as files or in the database
Software - ties all the above parts of the site together using a server-side language (PHP, etc.) as the controller - CMS, blog, image gallery, calendar, email app, etc.

Second, you need to understand how all these things are tied together. They work together as a "stack" of technologies--the "server" parts are foundational for the "website" parts, and the content of your site travels through the stack either from the server to the user, or from the user to the server. Here's the most common open-source "stack" out there--often called "LAMP" for "Linux, Apache, MySQL, PHP/Perl/Python."

LAMP
Model - MySQL - data is stored in a MySQL database
Controller - PHP, Apache - the site's software is written in PHP, which reads data from the database & creates an HTML page (inserting data where needed), gives the HTML to Apache, which sends it to the browser. Or, the PHP code takes data sent by a browser (from a form or a page's link to a URL) & decides what to do with it (maybe write it to the database or a file, send it in an email, etc.)
View - HTML, CSS, JavaScript - the browser interprets HTML (which is just a plain text file) &

renders/displays it as a web page. JavaScript or Flash are used to enrich the user interface here.

This stack is available on most web hosts, and can be set up on your home computer (via XAMPP) to learn PHP& MySQL or to make your programming go faster. If you don't want to write in a server-side programming language yet, you can create static HTML pages and test them on your home computer without installing/running web server software (Apache). Just double-click on the HTML file and you can view your site.

There are other stacks. Some are older and generally more complex, using languages whose syntax is verbose, hence time-consuming and error-prone:

- C or Perl used to be used as server-side languages
- Zope for the server, Python for the server-side language (simple), Plone for the CMS (complex), Zope Page Templates for the view (complex)
- Tomcat for the server & Java for the server-side language

Some are newer, and generally simpler and better. PHP is easy to learn, and would serve you well, but it's significantly easier to program in Python & Ruby. Ruby On Rails is a new stack that has popularized newer & better technologies and makes them easy to use. Right now I'm using a similar stack called TurboGears when I can.

TurboGears
Model - MySQL (or one of several other database backends it supports)
Controller - Python, CherryPy web server
View - XHTML templates, CSS, AJAX toolkits, widgets

3. Where to start

So, I'd recommend you do the following:

- Make a few static HTML pages to learn HTML. As needed, use them as a playground to learn CSS & JavaScript. Use the attached files to get started. You'll want to edit them in a text editor that gives you syntax highlighting - Notepad++ works well for Windows.
- If you want to learn web programming, install XAMPP & write some PHP pages, create a simple database using phpMyAdmin & use PHP to read from / write to it.
- If you want to learn AJAX, I have 3 simple example files I can send you.
- If you want to build a personal website, install & configure the Joomla CMS (add free plugins as needed) on a web server
- If you want to understand web 2.0 technology in general, get a free FaceBook profile and install a wiki on your web server (or just learn how to use someone else's wiki)
- If you need to understand web 2.0 programming, learn Ruby On Rails (the Rad Rails plugin in the Aptana IDE makes it easy), Django (in Python), or TurboGears

Let me know if you'd like me to give you more info or set up more examples on any of the above and I'll be glad to do so. If you want me to create a website for you, send me a list of the content & features you will want to have in the site, and I'll write a contract proposal you can modify or accept as the contract.

I wrote the following for another friend around 2010. It also is somewhat out of date now that applications are moving away from the three-tier architecture where the model and controller, and even the view generation code, runs on the server, to a two-tier architecture where nearly all the code except the database runs on the client.

Historical Background

First, be aware of some major changes that have happened in the history of web development. About every 5 years it seems to me web programming goes through a paradigm shift in the methods used to create a website and organize the programming code in a website. Each new paradigm incorporates some of the previous paradigm, but remains a major change. In 1990, most web pages were static (not interactive), but people handled form submissions using CGI scripts written in Perl. Around 1995, people switched to PHP & ASP, which did a better job of mixing HTML snippets and PHP or ASP (programming commands) code in the same file, but many programmers didn't organize their code well, so it was called "spaghetti code." Around 2000, people started organizing their code better. Around 2005/2006 JavaScript/AJAX libraries became more popular, and the current paradigm of using a "Rapid Web Application Framework" emerged, spearheaded by "Ruby on Rails." Since 2006 I've learned to use something very similar called TurboGears, but in the Python language rather than Ruby. Today things are shifting toward writing single-page applications (they don't refresh the page, but request new data via AJAX) for mobile smartphone touch screens that heavily depend on JavaScript interaction toolkits like Sencha Touch, jQuery Mobile and jQTouch.

Designer-friendly view/layout templates

Django is another Python rapid web application framework, and it was popularized with the story that it enabled its users to quickly modify the appearance of a busy news website (I think in Lawrence, KS). One key way it (and all rapid web application frameworks) made that possible was by organizing the code into a model (which interacts with the database), controllers (which move data from the page to the database or from the database to the page), and views/templates (which present the data in HTML which is then consumed by the web browser.) This is called the "MVC" (model-view-controller) design pattern. Rather than changing all three kinds of code to change the look of the page, as would be necessary with spaghetti code, often all that is necessary for quick layout changes is for a web designer who may be a non-programmer to modify an HTML view/template file.

Invest in newer, but popular and proven, programming languages and methods

So, though much of this may be new to you, I'd highly recommend avoiding investing in the older languages (Perl, PHP, ASP), or older methods (code not organized according to the MVC (or other similar) pattern(s)). It's true that web application frameworks move in and out of popularity quickly (say every 3-5 years?), yet because they encourage organizing code well, it is easier to migrate your code from an old framework to a new one piece by piece when it's organized well according to the MVC pattern. OSQA, for instance, is written in Django. Other software you may consider may be written in PHP or ASP. It can still be wise to pick software written in the older languages, because programming languages never really die--you'll be able to find Perl, PHP and ASP programmers for the next 20 years--but the newer languages Python and Ruby are overtaking the older ones in productivity and so popularity, and so may prove to be a better investment in the end. PHP and ASP web hosts are ubiquitous and cheap, and Python and Ruby web hosts (I use WebFaction, which used to be called Python-Hosting, and have found its tools and support to excel beyond other hosts I've used) are less common, but tend to be more technically competent because they are on the leading edge of developing technologies, and more of their staff and clients tend to be programmers building new software.

With this background, I'll mention why I chose TurboGears rather than Ruby on Rails, Django, CakePHP (in PHP), etc. Ruby is younger than Python, and has had some security problems in its language interpreter, and in 2006 had a less complete set of programming libraries to draw on to do specialized programming tasks. Python had a very complete set of libraries included by default (they say Python comes "batteries included.") Python and Ruby's syntax is simpler, cleaner, briefer, and more like plain English than PHP or ASP. ASP costs money to use because it's owned by Microsoft. Django is monolithic--you use only the Django model library, controller library, and view/template library. TurboGears breaks all functionality into swappable libraries--you can use whatever library you want, but TurboGears recommends a certain set of popular, sane and easy defaults. Because of a fundamental design problem in one of TurboGears' libraries named Pylons (which governs the controller code) that prevented Pylons from being developed further without breaking code that depends on older versions of Pylons, much of TurboGears' community/mindshare is migrating to using a new framework named Pyramid, which mostly plays the same role Pylons played (governing controller code; Pyramid is developed by the Pylons developers as the intentional replacement for Pylons), but also recommends using most of the same libraries used in TurboGears (SQLAlchemy for the model, and Genshi or Mako for the view templates) because they are among the most popular libraries in the Python world for those purposes, so it should be easier to migrate my code from TurboGears to Pyramid than it would be to move from Django to Pyramid.

Programmers' debates over which language or method is best are sometimes described as similar to wars over religion, with the implication that both are matters of preference wrongly turned into matters of life-or-death importance. I don't care what tool a person uses, but I have to choose tools to use myself. I mention the above to you so you can have some wisdom as you go into evaluating in what languages and methods a programmer may recommend you invest, and hopefully make a judgment about whether the investment is wise over the short and long term.

Some other links:

http://rubyonrails.org/
https://www.djangoproject.com/
http://turbogears.org/
https://www.pylonsproject.org/

I wrote the following in 2012, so it's up-to-date for now!

I formed the Caney Python User Group (CaneyPUGgies) to teach other people web programming, and make long-term contacts in this area beyond Caney. The CaneyPUGgies website describes how to get started learning with CaneyPUGgies, the tools we ARE using, and the programming languages and libraries we WERE using:

http://caneypuggies.alwaysreformed.com/

Our past meeting minutes show the process we've followed, with a detailed record of the code changes we've made:

http://caneypuggies.alwaysreformed.com/wiki/PastMeetingMinutes

We WERE writing our hands-on application " Reformed Churches Locator " in TurboGears , which is a rapid web application framework that integrates a large number of best-of-breed libraries and uses the Python language on the server side, and JavaScript on the client/browser side (as do all but Flash or Java web applications). Then we realized that one feature we need (bidirectional, filtered master-to-master data replication and versioning) would be difficult to write ourselves, so we moved to using the database called CouchDB since it provides that feature by default. This has required radically reorganizing our code, but also simplifies the code and allows us to reuse most of our JavaScript code. This is probably a good move, since web applications are increasingly being written in this new 2-layer way (single-page applications whose business logic runs in the browser in JavaScript and communicates to a document-oriented database on the server), rather than in the older three-layer way TurboGears (and the more popular Ruby on Rails) works (MVC - model-view-controller: model code communicates with the database, view code runs in the browser, controller code runs on the server and moves data between the model and the view.) So now we have moved to using the JavaScript language exclusively in both the browser and on the server, in CouchApps (a way of organizing the programming code) and CouchDB (a non-relational and document-oriented rather than relational database that runs on the server), and Node.js (an event-driven JavaScript interpreter that runs on the server, for when we need server-side code).

We need to update the CaneyPUGgies website to reflect the languages and libraries we are using now.

To get the tools you need to learn web development with us, you'll need to follow the instructions at http://caneypuggies.alwaysreformed.com/wiki/WebDevEnv. We can help you with all that at an upcoming meeting; see the meeting schedule at http://caneypuggies.alwaysreformed.com/wiki/PastMeetingMinutes.

Very practically, I recommend learning the basic languages first (see tutorials and cheat sheets at http://caneypuggies.alwaysreformed.com/wiki/CheatSheets), in this kind of order:

HTML/HTML 5
CSS
JavaScript
An overview of SQL and how to use a relational database like MySQL (which specific database you learn doesn't matter as much as does the general pattern of how to structure and query data in a relational database)
An overview of CouchDB to understand its different, non-relational paradigm for structuring and querying a database