



The World Wide Web: Crash Course Computer Science #30

Crash Course: Computer Science

<https://youtube.com/watch?v=guvsh5OFizE>

<https://nerdfighteria.info/v/guvsh5OFizE>

===== Introduction =====

(00:02) Hi, I'm Carrie Anne, and welcome to Crash Course Computer Science.

Over the past two episodes we've delved into the wires, signals, switches, packets, routers, and protocols that make up the internet. Today we're going to move up yet another level of abstraction and talk about the World Wide Web.

(00:16) This is not the same thing as the Internet, even though people often use the two terms interchangeably in everyday language. The World Wide Web runs on top of the internet, in the same way that Skype, Minecraft or Instagram do. The Internet is the underlying plumbing that conveys the data for all these different applications.

===== The World Wide Web =====

(00:31) And the World Wide Web is the biggest of them all: a huge distributed application running on millions of servers worldwide, accessed using a special program called a web browser. We're going to learn about that, and much more, in today's episode.

(00:52) The fundamental building block of the **World Wide Web**, or **Web** for short, is a single page. This is a document containing content which can include links to other pages. You all know what these look like: text or images that you can click and they jump you to another page.

(01:07) These **hyperlinks** form a huge web of interconnected information, which is where the whole thing gets its name. This seems like such an obvious idea, but before hyperlinks were implemented, every time you wanted to switch to another piece of information on a computer, you had to rummage through the file system to find it or type it into a search box. With hyperlinks, you can easily flow from one related topic to another.

(01:27) The value of hyperlinked information was conceptualized by **Vannevar Bush** way back in 1945. He published an article describing a hypothetical machine called a **Memex**, which we discussed in Episode 24. Bush described it as "**associative indexing**: whereby any item may be caused at will to select another immediately and automatically". He elaborated: "the process of tying two things together is the important thing... thereafter, at any time, when one of those items is in view, the other item can be instantly recalled merely by tapping a button".

(01:59) In 1945, computers didn't even have screens, so this idea was way ahead of its time. Text containing hyperlinks is so powerful, it got an equally awesome name: **hypertext**. Webpages are the most common type of hypertext document today. They're retrieved and rendered by **Web browsers**, which we'll get to in a few minutes.

===== Accessing Webpages =====

(02:15) In order for pages to link to one another, each hypertext page needs a unique address. On the Web, this is specified by a **Uniform Resource Locator**, or **URL** for short. An example webpage URL is "thecrashcourse.com/courses".

(02:29) Like we discussed last episode, when you request a site, the first thing your computer does is a DNS look-up. This takes a domain name as input, like "thecrashcourse.com", and replies back with the matching computer's IP address. Now armed with the IP address of the computer you want, your Web browser opens a TCP connection to a computer that's running a special piece of software

called a **Web server**. The standard port number for Web servers is **port 80**.

(02:51) At this point, all your computer has done is connect to the Web server at the address "thecrashcourse.com". The next step is to ask that Web server for the "courses" hypertext page. To do this, it uses the aptly named **Hypertext Transfer Protocol**, or **HTTP**.

(03:05) The very first documented version of this spec, HTTP version 0.9 created in 1991, only had one command: **GET**. Fortunately, that's pretty much all you need. Because we're trying to get the "courses" page, we send the server the following command: "GET /courses".

(03:20) This command is sent as raw ASCII text to the Web server, which then replies back with the webpage hypertext we requested. This is interpreted by your computer's Web browser and rendered to your screen. If the user follows a link to another page, the computer just issues another GET request, and this goes on and on as you surf around the website.

(03:38) In later versions, HTTP added status codes which prefixed any hypertext that was sent following a GET request. For example, status code 200 means "OK, I've got the page, and here it is." Status codes in the 400s are for client errors, like if the user asked the Web server for a page that doesn't exist. That's the dreaded 404 error!

(03:57) Webpage hypertext is stored and sent as plain old text, for example, encoded in ASCII or UTF-16, which we talked about in Episodes 4 and 20. Because plain text files don't have a way to specify what's a link and what's not, it was necessary to develop a way to mark up a text file with hypertext elements.

===== Creating Webpages =====

(04:12) For this, the **Hypertext Markup Language** was developed. The very first version of **HTML**, version 0.8 created in 1990, provided 18 HTML commands to mark up elements. That's it! Let's build a webpage with these.

(04:25) First, let's give our webpage a big heading. To do this, we type in the letters "h1", which indicates the start of a first-level **heading**, and we surround that in angle brackets. This is one example of an **HTML tag**. Then, we enter whatever heading we want. We don't want the whole page to be a heading, so we need to close the "h1" tag like so: with a little slash in the front.

(04:45) Now let's add some content. Visitors may not know what Klingons are, so let's make that word a hyperlink to the Klingon Language Institute for more information. We do this with an "a" tag, inside of which we include an attribute that specifies a **hyperlink reference** ("href"). That's the page to jump to if the link is clicked. And finally, we need to close the "a" tag.

(05:03) Now let's add a second-level heading, which uses an "h2" tag. HTML also provides tags to create lists. We start this by adding the tag for an **ordered list** ("ol"). Then we can add as many items as we want surrounded in "li" tags, which stands for **list item**. People may not know what a Bat'leth is, so let's make that hyperlink too. Lastly, for good form, we need to close the ordered list tag.

(05:24) And we're done! That's a very simple webpage. If you save this text into Notepad or Text Edit and name it something like "test.html", you should be able to open it by dragging it into your computer's Web browser.



The World Wide Web: Crash Course Computer Science #30

Crash Course: Computer Science

<https://youtube.com/watch?v=guvsH5OFizE>

<https://nerdfighteria.info/v/guvsH5OFizE>

(05:35) Of course, today's webpages are a tad more sophisticated. The newest version of HTML, version 5, has over a hundred different tags for things like images, tables, forms, and buttons. And there are other technologies we're not going to discuss, like **Cascading Style Sheets**, or **CSS**, and **JavaScript**, which can be embedded into HTML pages and do even fancier things.

===== Web Browsers =====

(05:53) That brings us back to Web browsers. This is the application on your computer that lets you talk with all of these Web servers. Browsers not only request pages and media, but also render the content that's being returned.

(06:04) The first Web browser and Web server was written by (now) **Sir Tim Berners-Lee** over the course of two months in 1990. At the time, he was working at CERN, in Switzerland. To pull this feat off, he simultaneously created several of the fundamental Web standards we discussed today: URLs, HTML, and HTTP. Not bad for two months work, although to be fair, he'd been researching hypertext systems for over a decade.

(06:27) After initially circulating the software among colleagues at CERN, it was released to the public in 1991. The World Wide Web was born! Importantly, the Web was an open standard, making it possible for anyone to develop new Web servers and browsers.

(06:39) This allowed a team at the University of Illinois, at Urbana-Champaign, to create the Mosaic Web browser in 1993. It was the first browser that allowed graphics to be embedded alongside text. Previous browsers displayed graphics in separate windows. It also introduced new features, like bookmarks, and had a friendly GUI interface, which made it popular. Even though it looks pretty crusty, it is recognizable as the Web we know today.

(07:00) By the end of the 1990s there were many Web browsers in use, like Netscape Navigator, Internet Explorer, Opera, OmniWeb, and Mozilla. Many Web servers were also developed, like Apache and Microsoft's Information Services (IIS). New websites popped up daily, and Web mainstays like Amazon and eBay were founded in the mid-1990s. It was a golden era!

===== Search Engines =====

(07:20) The Web was flourishing, and people increasingly needed ways to find things. If you knew the Web address of where you wanted to go, like "ebay.com", you could just type it into the browser. But what if you didn't know where to go? Like you only knew you wanted pictures of cute cats. *Right now*. Where do you go?

(07:34) At first, people maintained webpages which served as directories hyper-linking to other websites. Most famous among these was Jerry and David's Guides to the World Wide Web, renamed Yahoo in 1994. As the Web grew, these human-edited directories started to get unwieldy, and so **search engines** were developed. Let's go to the Thought Bubble!

===== Thought Bubble: Search Engines =====

(07:52) The first Web search engine that operated like the ones we use today was JumpStation, created by Jonathon Fletcher in 1993

at the University of Sterling. This consisted of three pieces of software that worked together.

(08:03) The first was a **web crawler**: software that followed all the links it could find on the web. Any time it found a link on a page that had new links, it would add those to its list. The second component was an ever-enlargening index, recording what terms appeared on what pages the crawler had visited. The final piece was a search algorithm that consulted the index.

(08:21) For example, if I type the word "cat" into JumpStation, every webpage where the word "cat" appeared would come up in the list. Early search engines used very simple metrics to rank-order the search results; most often, [this was] just the number of times the search term appeared on a page. This worked okay until people started gaming the system, like by writing "cat" hundreds of times on their webpages just to steer traffic their way.

(08:43) Google's rise to fame was, in large part, due to a clever algorithm that sidestepped this issue. Instead of trusting the content on a webpage, they looked at how other websites linked to that page. If it was a spam page with the word "cats" written over and over again, no site would link to it. But if the webpage was an authority on cats, then other sites would likely link to it. So the number of what are called **backlinks**, especially from reputable sites, was often a good sign of quality.

(09:07) This started as a research project called **BackRub** at Stanford University in 1996 before being spun out two years later into the Google we know today. Thanks Thought Bubble!

===== Net Neutrality =====

(09:17) Finally, I want to take a second to talk about a term you've probably heard a lot recently: **net neutrality**. Now that you've built an understanding of packets, Internet routing, and the World Wide Web, you know enough to understand the essence, or at least the technical essence, of this big debate. In short, network neutrality is the principle that all packets on the Internet should be treated equally. It doesn't matter if the packets are my email or you streaming this video; they should all chug along at the same speed and priority.

(09:41) But *many* companies would prefer that their data arrived to you preferentially. Take, for example, Comcast: a large ISP that also owns many TV channels, like NBC and The Weather Channel, which are streamed online. Not to pick on Comcast, but in the absence of net neutrality rules, they could, for example, say that they want their content delivered silky-smooth and high-priority, but other streaming videos are going to get throttled; that is, intentionally given less bandwidth and lower priority. Again, I want to reiterate here: this is just conjecture.

(10:09) At a high level, net neutrality advocates argue that giving Internet providers this ability, to essentially set up tolls on the Internet, to provide premium packet delivery, plants the seeds for an exploitative business model. ISPs could be gatekeepers to content with strong incentives to not play nice with competitors. Also, if big companies like Netflix and Google can *pay* to get special treatment, small companies like startups will be at a disadvantage, stifling innovation.

(10:33) On the other hand, there are other reasons you might want different types of data to flow at different speeds. That Skype call needs high priority, but it's not a big deal if that email comes in a few seconds late. Net neutrality opponents also argue that market forces and competition would discourage bad behaviour because



The World Wide Web: Crash Course Computer Science #30

Crash Course: Computer Science

<https://youtube.com/watch?v=guvsh5OFizE>

<https://nerdfighteria.info/v/guvsh5OFizE>

customers would leave ISPs that are throttling sites they like.

(10:53) This debate will rage on for a while yet, and as we always encourage on Crash Course, you should go out and learn more, because the implications of net neutrality are complex and wide-reaching. I'll see you next week.

===== Credits =====

(11:04) Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like Brain Craft, Coma Niddy, and PBS Infinite Series. This episode was filmed at the Chad and Stacey Emigholz Studio in Indianapolis, Indiana, and it was made with the help of all of these nice people and our wonderful graphics team, Thought Cafe. Thanks for the random access memories. I'll see you next time.