RÉFÉRENCE HTML

<https://developer.mozilla.org/en-US/docs/Learn/HTML>

Open your command prompt (Windows)/terminal (OS X/Linux). To check Python is installed, enter the following command:

python -V

  This should return a version number. If this is OK, navigate to the directory that your example is inside, using the cd command.

# include the directory name to enter it, for example

cd Desktop

# use two dots to jump up one directory level if you need to

cd ..

  Enter the command to start up the server in that directory:

# On Mac and Linux

python -m SimpleHTTPServer

# On Windows

python -m http.server

  By default, this will run the contents of the directory on a local web server, on port 8000. You can go to this server by going to the URL localhost:8000 in your web browser. Here you'll see the contents of the directory listed — click the HTML file you want to run.

**Note**: If you already have something running on port 8000, you can choose another port by running the server command followed by an alternative port number, e.g. python -m SimpleHTTPServer 7800. You can then access your content at localhost:7800.

-----------------------

**Running server-side languages locally**

Python's SimpleHTTPServer module is useful, but it doesn't know how to run code written in languages such as PHP or Python. To handle that you'll need something more — exactly what you'll need depends on the server-side language you are trying to run. Here are a few examples:

* To run Python server-side code, you'll need to use a Python web framework. You can find out how to use the Django framework by reading [Django Web Framework (Python)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django). [Flask](http://flask.pocoo.org/) is also a good (slightly less heavyweight) alternative to Django. To run this you'll need to [install Python/PIP](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment#Installing_Python_3), then install Flask using pip3 install flask. At this point you should be able to run the Python Flask examples using for example python3 python-example.py, then navigating to localhost:5000 in your browser.
* To run Node.js (JavaScript) server-side code, you'll need to use raw node or a framework built on top of it. Express is a good choice — see [Express Web Framework (Node.js/JavaScript)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs).
* To run PHP server-side code, you'll need a server setup that can interpret PHP. Good options for local PHP testing are [MAMP](https://www.mamp.info/en/downloads/) (Mac and Windows) , [AMPPS](http://ampps.com/download) (Mac, Windows, Linux) and [LAMP](https://www.linux.com/learn/easy-lamp-server-installation) (Linux, Apache, MySQL, and PHP/Python/Perl). These are complete packages that create local setups to allow you to run the Apache server, PHP, and MySQL databases.

------------

To choose a color, go to [the Color Picker](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Colors/Color_picker_tool)

#33DAF4

-------------

image: <https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjuhbXArvPVAhWL8YMKHSUlBDQQjRwIBw&url=http%3A%2F%2Ftampabaybeersyndicate.com%2F&psig=AFQjCNGlPuB8eS2pMp_ZfYhT6hWKEGDBzw&ust=1503784071426593>

-----------------

Font: Google fonts

**Embed Font**

To embed your selected fonts into a webpage, copy this code into the <head> of your HTML document.

<link href="https://fonts.googleapis.com/css?family=Droid+Sans" rel="stylesheet">

**Specify in CSS**

Use the following CSS rules to specify these families:

font-family: 'Droid Sans', sans-serif;

--------------------------

Some general rules for file paths:

* To link to a target file in the same directory as the invoking HTML file, just use the filename, e.g. my-image.jpg.
* To reference a file in a subdirectory, write the directory name in front of the path, plus a forward slash, e.g. subdirectory/my-image.jpg.
* To link to a target file in the directory **above** the invoking HTML file, write two dots. So for example, if index.html was inside a subfolder of test-site and my-image.jpg was inside test-site, you could reference my-image.jpg from index.html using ../my-image.jpg.
* You can combine these as much as you like, for example ../subdirectory/another-subdirectory/my-image.jpg.

----------------------



Attributes contain extra information about the element that you don't want to appear in the actual content. Here, class is the attribute name, and editor-note is the attribute value. The class attribute allows you to give the element an identifier that can be later used to target the element with style information and other things.

An attribute should always have:

1. A space between it and the element name (or the previous attribute, if the element already has one or more attributes).
2. The attribute name, followed by an equals sign.
3. Opening and closing quote marks wrapped around the attribute value.

--------------------------

<!DOCTYPE html> — the doctype. In the mists of time, when HTML was young (about 1991/2), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However, these days no one really cares about them, and they are really just a historical artifact that needs to be included for everything to work right. For now, that's all you need to know.

<html></html> — the [<html>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html) element. This element wraps all the content on the entire page, and is sometimes known as the root element.

<head></head> — the [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head) element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like [keywords](https://developer.mozilla.org/en-US/docs/Glossary/keyword) and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.

<body></body> — the [<body>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body) element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

<meta charset="utf-8"> — this element sets the character set your document should use to UTF-8, which includes most characters from the vast majority of human written languages. Essentially it can now handle any textual content you might put on it. There is no reason not to set this, and it can help avoid some problems later on.

<title></title> — the [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title) element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favorite it.

--------------

Anything in a CSS document between /\* and \*/ is a **CSS comment**