Module 1

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 1 Study Guide and Deliverables

Readings:	Brown: Chapters 1–3
Assignments:	Assignment 1 due Tuesday, January 24 at 6:00 AM ET
Live Classroom:	• Tuesday, January 17, 6:00–8:00 PM ET • Facilitator live office: TBD

Survey of Server-Side Languages

Learning Objectives

By reading the lectures and textbook, participating in the discussions, and completing the assignments, you will be able to do the following:

- Identify key Server-Side programming languages.
- Understand some of the history behind server-side programming.

Introduction

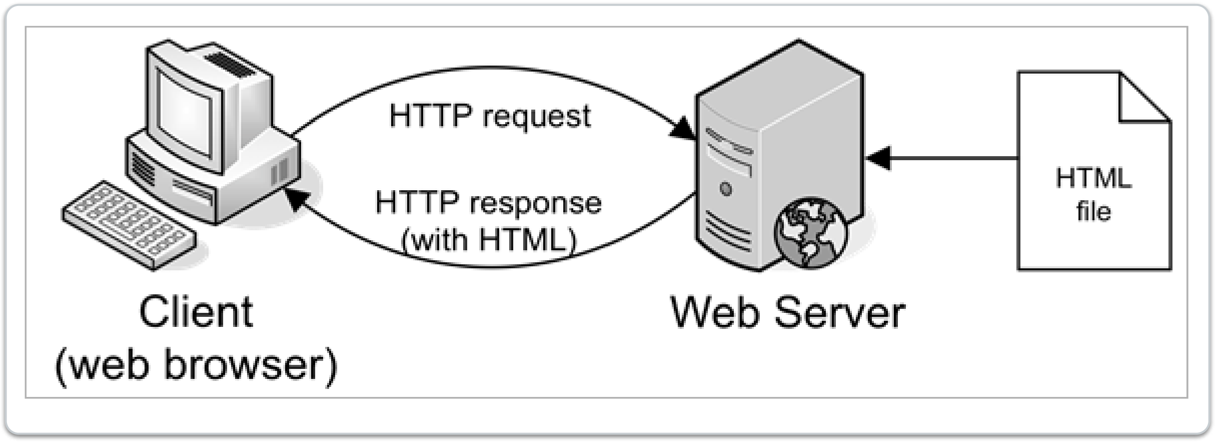
The focus of this course is on server-side web development. We will create applications that respond to the dynamic needs of each web visitor. Server-side scripts are different from client-side scripts such as JavaScript, which are run in the web browser.

Server-side programming has been around for a long time. PHP is 20 years old and other server-side scripting languages have been around for a very long time as well. Before we had languages like PHP we were able to use the Common Gateway Interface (CGI) to produce similar programs, but with limited functionality. CGI is an environment for web servers to interface with executable programs installed on the server. This allowed developers to use languages such as C, Perl, and even shell scripts in order to generate web pages in a dynamic way.

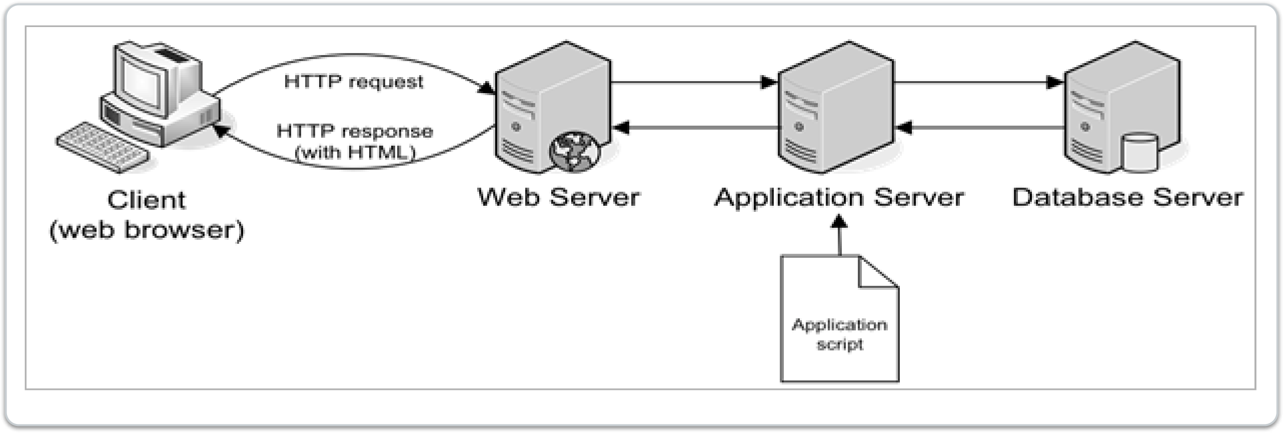
In this lecture, we will outline the request-response cycle and introduce some of the more popular server-side languages.

Request-Response Cycle

The traditional request-response cycle when dealing with client-side technologies like HTML, CSS, and JavaScript is fairly simple and doesn't require a lot of coordinated server activities.



However, when we start using server-side languages and a database, the process becomes a bit more involved.



You can see in the second diagram, the web server is interacting with an application server (like PHP), which in turn works with a database sever to retrieve the requested data and all of this gets sent back to the web browser as HTML.

Server-Side Languages

We use server-side languages to create web applications and services. A full web application experience consists of four major components:

- Web browser
- Web server
- Server-side language(s)
- Database server(s)

There are many, many different programming languages that can be used to develop server-side applications. In fact, there are so many different ones that it can often make it difficult to choose which ones to use for development projects. In this course, we will be using PHP and Node.js to develop our own development projects and they will be our focus throughout the next six weeks of this course.

We are going to spend a little bit of time below providing an overview of today's most widely used server-side languages.

PHP

PHP was created in 1995 specifically for use in web development but it can also be used as a general purpose programming language. PHP is installed on approximately 70% of all web servers in the world and powers hundreds of millions of websites. PHP code can easily be mixed in with HTML code. The PHP code contained within web pages is parsed by the PHP interpreter residing on the server and then sends the output back as HTML to the web browser. PHP supports procedural and object oriented programing paradigms.

The PHP interpreter is powered by the Zend Engine and is free software released under it's own PHP license. It is available on nearly every operating system and platform. The current stable version of PHP is version 5.6 and version 7.0 is currently in development with a beta version released in July 2015.

You can learn more about PHP on its home page, located at http://php.net

ASP.NET

ASP.NET is an open-source server-side web application framework developed by Microsoft and designed for web development in order to produce dynamic web applications and services. Version 1.0 was released in 2002, which replaced Microsoft Active Server pages. As of July 2015, the latest version is 4.5. Web pages created in ASP.NET are known as Web Forms, which are the building blocks of a web application.

You can learn more about ASP.NET on its home page.

Python

Python is a general-purpose, high-level programming language. It focuses on code readability and is very popular for teaching programming to new comers. It was conceived in the late 1980's by Guido van Rossum and it is open-source software. Python can be extremely suitable for developing web applications using some of these popular frameworks:

1. Djang
2. Pyramid
3. Bottle
4. Tornado
5. Flask
6. Web2py

Learn more about Python.

Perl

Perl is a highly capable, feature-rich programming language with over 27 years of development and over 108,000 modules available. It is currently on version 5 with version 6 under development. Perl has been used to write CGI scripts. Perl is not as popular as it used to be for web development but it is still actively used today more than most people would assume. Large projects written in Perl include cPanel, BugZilla, and Moveable Type. There are also a number of popular websites that use Perl such as Priceline, Craigslist, IMDb, Ticketmaster, Slashdot, and DuckDuckGo.

Perl is an optional component of the LAMP stack (Linux, Apache, MySQL, and PHP) where it is sometimes used instead of PHP.

Learn more about Perl.

JSP

JavaServer Pages (JSP) was released by Sun Microsystems in 1999 and it allows developers to create dynamic web pages based on HTML and other document types. It is similar to PHP, but it is implemented with the Java programming language. In order to deploy and run JSP, a special web server such as Apache Tomcat is required. JSP, like Java, is maintained by Oracle today.

Learn more about JSP.

Ruby on Rails

Ruby on Rails (Rails) is a web application framework that is written using the Ruby programming language. Ruby is an object oriented, general-purpose programming language developed in the 1990's by Yukihiro "Matz" Matsumoto. Rails was developed by David Heinemeier Hansson in 2004. It is a very popular web framework that has grown rapidly in the past decade, with much of that growth slowing during the last couple of years. It is a model-view-controller (MVC) framework and emphasizes the use of well-known software engineering patterns. The current version, 4.2, was released in December 2014.

Learn more about Rails and more information can be found on the Ruby Programming language.

JavaScript

Really? JavaScript - the client-side programming language? ***Yes!***

Even though JavaScript is a very popular client-side language, it can be used to develop very powerful and efficient server-side applications. In order to do so, we need a runtime environment specifically designed to allow JavaScript to be used on the server. Node.js is that runtime environment.

Node.js is open-source and cross platform. It is event driven and non-blocking that allows for high throughput and impressive scalability. It is rapidly growing in popularity as a server-side development environment and has been used by Walmart, LinkedIn, PayPal, Microsoft, Yahoo and many other large organizations that rely on high performance web applications.

Learn more about Node.js.

Summary

In this lecture, we learned some of the history of server-side web development and an overview of today's more popular server-side programming languages to include PHP, ASP.NET, Python, Perl, JSP, Ruby on Rails, and JavaScript with Node.js.

In the next module, we are going to take a deep dive into programming with PHP and use it to create dynamic web sites in conjunction with a MySQL database.

Bibliography

The PHP Group. (2015). *PHP Home Page*. Retrieved July 15, 2015, from http://php.net

Microsoft. (2015). *ASP.NET Home Page*. Retrieved July 15, 2015, from http://www.asp.net

Python Software Foundation. (2015). *Python Home Page*. Retrieved July 15, 2015, from https://www.python.org

Perl.org. (2015). *The Perl Programming Language*. Retrieved July 15, 2015, from https://www.perl.org

Oracle. (2015). *JavaServer Pages Technology*. Retrieved July 15, 2015, from http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Members of the Ruby Community. (2015). *Ruby Home Page*. Retrieved July 15, 2015, from https://www.ruby-lang.org/en

Hansson, D. H. (2015). *Ruby on Rails Home Page*. Retrieved July 15, 2015, from http://rubyonrails.org

Node.js Foundation. (2015). *Node.js Home Page*. Retrieved July 15, 2015, from http://nodejs.org

Core Node.js

Learning Objectives

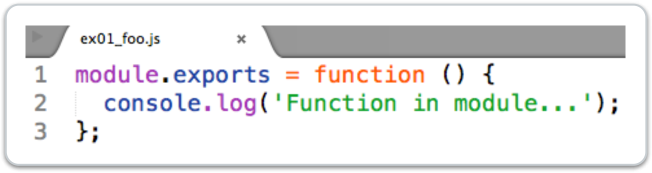
By reading the lectures and textbook, participating in the discussions, and completing the assignments, you will be able to do the following:

- Describe the structure of Node applications.
- Develop module based applications for reusable code.
- Use the core Node.js modules.
- Examine how the node package manager works.
- Install third-party node modules as part of your applications.

Introduction

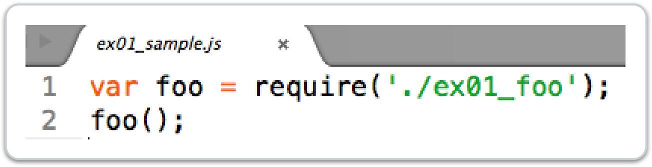
Node.js is based on the concept of modules and follows the CommonJS module specification. The JavaScript code that is executed on the server-side is organized into various modules. In the module-based approach, each JavaScript file is its own module. In each file, the module variable provides access to the current module definition. The functionality exported by the module is determined by the module.exports variable. Applications that require the module's functionality use the require function to import the module.

The following example shows a module that exports a single function.



```javascript
module.exports = function () {
  console.log('Function in module...');
};
```

In the main Node application, the module is imported into the variable, foo. Since the module is exporting a single function, the function is then invoked as shown below.



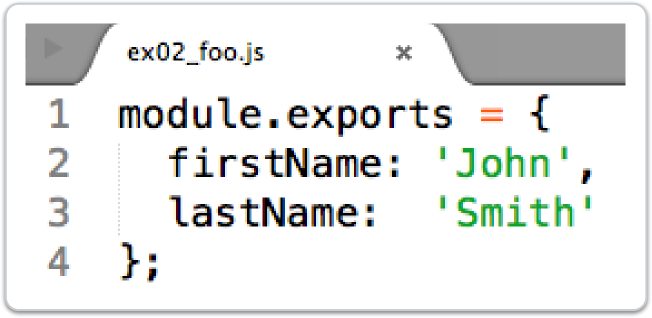
```javascript
var foo = require('./ex01_foo');
foo();
```

The output of the above application is shown below.

```
>node ex01_sample.js
Function in module...
```

Exporting Objects and Sharing State

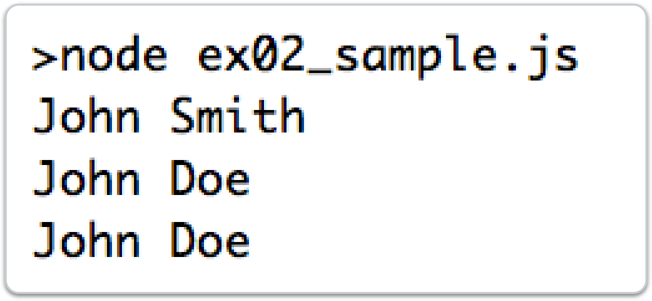
When a module is loaded by the require function, the module definition is cached and subsequent require calls within the application return the cached module definition. This allows the state to be shared between modules. The following example shows a JavaScript object being exported by the module.



```javascript
module.exports = {
  firstName: 'John',
  lastName:  'Smith'
};
```

In the main application, the module is first loaded into the variable, foo1. After accessing the firstName and lastName properties, the lastName value is changed. When the module is again loaded into a different variable, foo2, the changed state is visible. In a practical application, the loading of the same module will be in separate files and they will be able to share the module’s state.

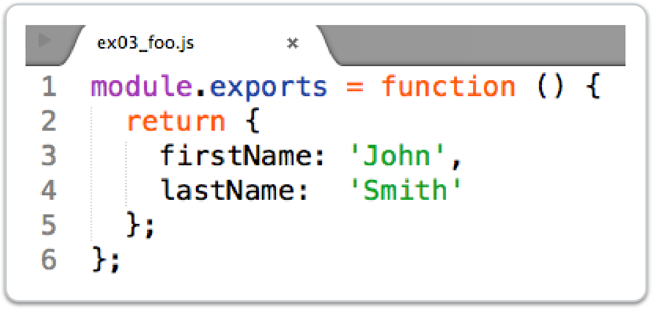
The output of the above program is shown below.



```
>node ex02_sample.js
John Smith
John Doe
John Doe
```

Using Object Factories

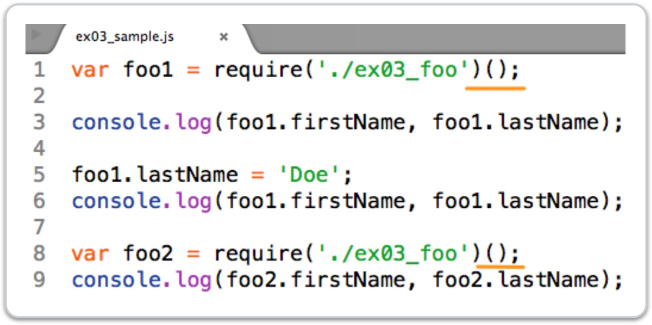
In the previous example, the same object is returned for multiple require calls. However, if a new object is required for each require call, then the module can export a function rather than the object. Invoking the exported function would then return a new object. The following example exports a single function that returns the JavaScript object, when invoked.



ex03_foo.js

```javascript
module.exports = function () {
  return {
    firstName: 'John',
    lastName:  'Smith'
  };
};
```

In the main application, the module is loaded and then invoked to get the reference of the data into the variable, foo1. Any changes to the state of this data is only visible through the variable, foo1. If the module is loaded again and then invoked, a new object is returned.



ex03_sample.js

```javascript
var foo1 = require('./ex03_foo')();

console.log(foo1.firstName, foo1.lastName);

foo1.lastName = 'Doe';
console.log(foo1.firstName, foo1.lastName);

var foo2 = require('./ex03_foo')();
console.log(foo2.firstName, foo2.lastName);
```

The output of the above program is shown below.

```
>node ex03_sample.js
John Smith
John Doe
John Smith
```

Module Exports

A typical module exports more than a single object or a single function. The following example shows the first approach of how the functionality is exported by the module. Aliases for the function definitions are used in the module exports.

The main application uses the above module as shown below. The data from the module is not directly accessible to the application. The application can only access the data indirectly through the properties exported by the module.

The output of the above application is shown below.

A second approach is to define the functions along with the properties of the exported JavaScript object as shown below.

The main application uses the above module as shown below. The data from the module is not directly accessible to the application. The application can only access the data indirectly through the properties exported by the module.

The output of the above application is shown below.

A third approach used in defining the modules is shown below. The properties are directly set using the global module.exports, or the alias for it, the global exports variable.

The main application uses the above module as shown below. The data from the module is not directly accessible to the application. The application can only access the data indirectly through the properties exported by the module.

The output of the above application is shown below.

Node.js Globals

Node.js provides a good number of global variables that can be used in applications. The global variable is a JavaScript object that is first looked up for global property references. The console is one these properties.

The __dirname and __filename variables are available in each file and give the full path of the current directory and the full path of the current file. The setInterval and clearInterval properties are used to specify code that needs to executed repeatedly at the specified frequency and to stop the execution. The process property returns the process object whose argv member property can be used to access the command line arguments. The following example illustrates some of the global variables.

The output of the above application invoked with the specified arguments is shown below.

Node.js Core Modules

Node.js provides a variety of functionality that can be reused across various applications. The name of the module is specified for the require function in order access the functionality exported by that module. The following sections show the usage of some of the core modules.

path Module

The path module provides the functions for handling and transforming file paths. The file system itself is not consulted for validity of the files. The following example shows the usage of the functions dirname, basename, and extname from the module. The dirname method returns the directory name of the specified path. The basename method returns the last portion of the specified path. The extname method returns the extension of the specified path, or the empty string if no extension is there.

fs module

The fs module provides access to the file system. The module exports functions for reading files, writing files, deleting files, and renaming files. Most of the operations have both synchronous and asynchronous options.

The following example shows the synchronous versions of writing content to a file (writeFileSync) and reading content from the file (readFileSync). By default, the data read is returned as a Buffer object. The toString method converts the buffer to the corresponding string. The optional encoding value can also be specified to read the content directly as a string.

The output of the above program is shown below.

The asynchronous version of reading the file using the readFile function is shown below. The callback function is specified as the last argument of this function. When the read is complete, or if there is an error, the callback is called with the corresponding arguments. On successful read, the second argument of the callback function contains the contents read from the file.

The output of the above program is shown below.

The asynchronous version of writing to the file using the writeFile function is shown below. The callback function is specified as the last argument of this function. When the write is complete, or if there is an error, the callback is called as shown below.

The output of the above program is shown below.

The synchronous version of deleting the file using the unlinkSync function is shown below.

The output of the above program when the file exists is shown below.

The output of the above program when the file does not exist is shown below.

The asynchronous version of deleting the file using the unlink function is shown below. The callback function is called when the delete is successful, or when an error occurs.

The output of the above program when the file exists is shown below.

os module

The os module provides operating system related utility properties and functions. The following example shows the total memory (totalmem) and the number of CPUs (cpus) of the current system.

```
.
```

The output of the above program is shown below.

```
.
```

Boston University Metropolitan College