

Lecture Week 7-1: Web Scripting

Scripting Languages Introduction
Web Scripting Best Practices
Client-side and server-side scripting

Scripting Languages Introduction

Originally a scripting language or script language was a programming language to write programs that can **interpret** and **automate** the execution of other applications. Web pages within a web browser are a good case of applications that can be automated through scripting.

In web development, scripts are computer programs that allow web pages to be interactive and dynamic based on user input. Scripts can be categorised as either client-side or server-side, depending on where they are executed.

A scripting language was designed to **glue** together other applications and tools. Thus, you can reuse existing applications to create your own application or, you can get existing applications to act the way you want; e.g. Using JavaScript you can change the behaviour of browsers.

Typically, a scripting language is characterised by the following properties:

Ease of use:

Scripting languages are intended to be very fast to pick up and author programs in. This generally implies relatively simple syntax and semantics.

Database and file-system facilities:

Scripting is built in with easy interfaces to use databases and the file system.

Interpreted from source code:

To give the fastest turnaround from script to execution; so scripts can be written and executed without explicit compile and link steps.


Relatively loose structure:

It would be difficult to use *Java* as a scripting language due to the rules about which classes exist in which files - contrast to *Python*, where it's possible to simply define some functions in a file.

In truth, the term *scripting language* is a somewhat awkward one now because many of the modern scripting languages have outgrown the group's scripting origins and are now standalone general-purpose programming languages of considerable power (e.g. *Python*). We continue using it because nobody has yet invented a better term.

A Software Engineer specialised in Web Programming should be comfortable mixing various scripting and general-purpose languages within large web-based applications. What makes scripting languages stand out from general-purpose programming languages is that they have a **runtime to do interpretation** and to **automate dynamic storage management**.

Web Scripting “Rules”

- Rule of Modularity: Write simple scripts connected by clean interfaces.
- Rule of Clarity: Clarity is better than cleverness. Rewrite any confusing code.
- Rule of Composition: Design programs to be connected with other programs.
- Rule of Separation: Separate design from functionality; separate user interfaces from data source engines.
- Rule of Simplicity: Design for simplicity; add complexity only where you must.
- Rule of Minimum Size: Keep programs very small and manageable. Write a big program only when it is clear by demonstration that nothing else will do.
- Rule of Transparency: Program for visibility to make inspection and debugging easier (not only indentations).
- Rule of Robustness: Robustness is the child of transparency and simplicity.
- Rule of Representation: Fold knowledge into data, so program logic can be stupid and robust. Data structures, not algorithms, are central to programming.
- Rule of Least Surprise: In interface design, always do the least surprising thing. + means *plus (add)* OK?
- Rule of Repair: Cope nicely with errors that you can detect - but when you must fail, fail noisily and as soon as possible.
- Rule of Economy: Programmer time is expensive; conserve it in preference to machine time. Automated as much as you can.
- Rule of Production-first: Avoid hand-hacking programmes that are in production; use a development server instead.
- Rule of Testing: Prototype before polishing. Get it working before you optimize it. Your best friend is your end user.
- Rule of Optimization: Distrust all claims for one true way 
- Rule of Extensibility: Design for the future, because it will be here sooner than you think.
- Rule of Universality: Always create accessible web content for everyone

(Adapted from "The Art of Unix Programming" by Eric Steven Raymond)

Safely reuse others' code as much as you can. The idea that new code is better than old is patently absurd. Old code has been used. It has been tested. Lots of bugs have been found, and they've been fixed.

In other words:

Write programs that do one thing and do it well.

Write programs to work together.

Write programs that can be read by humans.

Be tough in what you accept, and correct in what you give

Reap the benefits of collaborative development

Apply defensive programming techniques to reduce and flush out errors

Resolve critical construction issues early and correctly

Build quality into the beginning, middle, and end of your project

Think in maintenance. It's harder to read code than to write it.

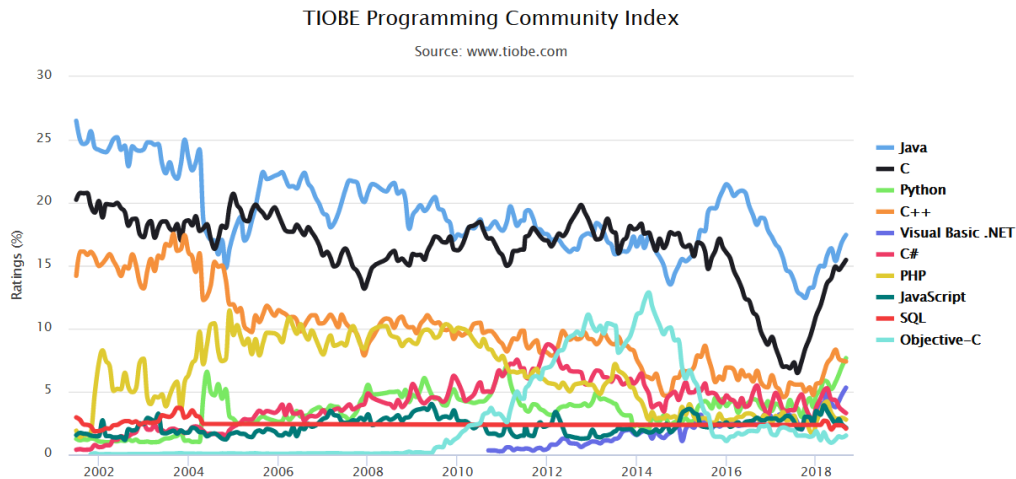
Client-side and server-side scripting languages

There isn't a clear line drawn between client-side and server-side scripting languages. The main distinction between the two is the where about the execution of the code takes place. Some other distinctions include that server-side scripting gives more control (security) and it is more powerful programmability than client-side scripting. Server-side scripts are hidden (the user will only be able to see the *HTML* output even if they attempt to get access to the source.) It is important to know to choose between using client or server-side scripting when developing web applications. You are also be expected to use them simultaneously.

JavaScript is a kind of special case of client-side script language. It is possible to write server-side scripts with JavaScript (Server-Side JavaScript (SSJS) is an extended version of JavaScript that enables back-end access to databases, file systems, and server's applications). In theory, using JavaScript as a server-side scripting language would allow you to exploit it in its entirety by building more scalable, event-driven and non-I/O blocking web applications. One example of server-side JavaScript environment is Node.js. PHP is a typical server-side scripting language used in web development.

Client-side scripts are executed on the users' device (e.g. web browser.) They allow for the creation of faster and more responsive web applications. *JavaScript* is the most popular client-side scripting languages. These scripts can be **embedded** (when the script is written directly within *HTML* code, contained between `<script>` and `</script>` tags) or **external** (when the script is in separated file that is called by the *HTML* using the filename provided in the `src` attribute of the `<script>` tag.) *JavaScript* (*JS*) can be used to develop complex problems. Thus, *JS* uses patterns, object-oriented (OO) coding, *AJAX*, etc. Advanced *JS* developers have created JavaScript libraries, which are pre-written JavaScript code to enable easier development of JavaScript-based applications (e.g. *AngularJs*, *ExpressJs*, etc.) The most popular *JS* library is probably *jQuery*. One popular scripting language in the iOS environment is *Objective-C*, which is widely used to develop mobile apps. To execute, the scripting language needs to be enabled on the client computer.

Server-side scripts are code that is executed (interpreted) on the web server, before the web page is sent to the browser, in response to requests received from a client-side application (e.g. web browser). The purpose of server side scripts is to implement the business logic of web applications, by processing user inputs, updating databases and data sources and, interacting with other applications residing on the server. There are various server-side scripting languages. The decision of choosing a server-side language should be based on the security, architecture, scalability, performance and functional requirements of the problem as well as using web programming best practices. By comparison, syntax is the less important part. However, it is advisable to have same level of expertise in the most demanded scripting languages such as *Python*, *PHP* and *Ruby*.



Some big changes to web application architecture since 3-tier includes:

Web Service Oriented Architecture, in which a lot of the code for the full web application exists as services, each with its own Web API. So when service a website, one part of the full web application (one web API) can make a request to another part of the code (a second API), often running on a different web server or on the cloud. The APIs are frequently organised according to REST design principles. The request and the response are typically just data, not necessarily human-friendly representations.

Single-page application, in which important part of the web application functionality is provided by a rich JavaScript code; which stays in the user's browser over many user interactions. The UI can perform many asynchronous or synchronous requests to the web server without reloading the webpage. It uses AJAX or WebSockets for this communication. This gives the user a smoother more natural experience, without being interrupted by page loads.

Together these two trends have enabled web apps to be designed for multiple platforms. This is important in the modern world when many users are on tablets or smartphones devices, instead of full-screen browsers on laptops or desktop computers. These technologies, together with RWD and AWD techniques, enable the design of websites that can fit smaller screens, even while most of the code for the full web application remains the same.

Furthermore, the above two trends have motivated and at the same time, been favoured by the development of modern front-end and back-end development frameworks, such as Angular, React, Redux, Flux, Ruby on Rails, Django , Node.js with express and, many more.

Definition of N-Tier Architecture

N-tier architecture is also called multi-tier architecture because the software is engineered to have the processing, data management, and presentation functions physically and logically separated. That means that these different functions are hosted on several machines or clusters, ensuring that services are provided without resources being shared and, as such, these services are delivered at top capacity. The "N" in the name n-tier architecture refers to any number from one.

Not only does your software gain from being able to get services at the best possible rate, but also it is also easier to manage. This is because when you work on one section, the changes you make will not affect the other functions. Moreover, if there is a problem, you can easily pinpoint where it originates. Multi-tier architecture enables the management of high volume of data transactions from different database systems.

Cloud Computing Terminology

A **Fabric Module** is bus network system that allows a Cloud System to scale, as your bandwidth needs increase because each fabric module added increases the total amount of throughput of the overall system.

The **Server Module** is the CPU of the cloud system. It is a farm of virtual servers (VMs) sharing resources of physical servers.

The **Storage Module** provides data storage for the cloud computer. It comprises the SAN (Storage-Area-Network) and the storage subsystem that connects storage devices.

Further Reading

Code Complete: A Practical Handbook of Software Construction by Steve McConnell

Software Engineering Best Practices by Capers Jones

The Developer's Code: What Real Programmers Do by Ka Wai Cheung

PHP and MySQL Web Development by Luke Welling and Laura Thomson

<http://codebalance.blogspot.co.uk/2011/02/20-software-developing-best-practices.html>

<http://hyperpolyglot.org/scripting>

<http://www.nczonline.net/blog/2012/06/19/web-developers-are-software-engineers-too/>

<https://www.creativeblog.com/netmag/coding-efficiency-beginners-write-reusable-code-1126513>

<https://developers.google.com/web/fundamentals/>

<http://best-practice-software-engineering.ifs.tuwien.ac.at/>

<https://www.sciencedirect.com/book/9780123849199/private-cloud-computing>

<http://highscalability.com/>

<http://programmers.stackexchange.com/questions/171203/what-are-the-difference-between-server-side-and-client-side-programming>

<https://docs.python.org/2/howto/webrowsers.html>

<http://www.phpthewrongway.com/>

<http://codersatwork.com/>

<http://www.defprogramming.com/>