Introduction to open-Source Software (OSS)

Concepts, strategies, and methodologies related to open-source software development

Week 10 – Lecture 08

Jamil Hussain
jamil@sejong.ac.kr
010-6252-8807

Office: 421, Innovation Center
Sejong University

Recap

◯ Markdown Language overview

◯ Markdown – Application

◯ Markdown - Basic Syntax

- List
- Image
- URL etc.

세종대학교
SEJONG UNIVERSITY

Today, Agenda

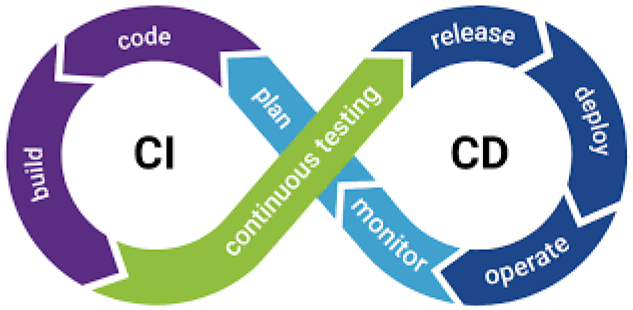
- Why Learn YAML?
- What is YAML?
- YAML - Comments
- Basic Syntax
- YAML - Array & collection
- YAML - Multiline Strings

- Jekyll
  - Installation
  - Getting started with Jekyll
  - Introduction to Jekyll front matter & YAML
  - Posts
  - Pages
  - Permalinks
  - Collection
  - Data files
  - Liquid Templating Framework
  - Layouts

세종대학교 SEJONG UNIVERSITY

YAML is Popular

- Almost all the popular configurations files are written in YAML
- Widely used format
  - for different DevsOps tools and applications



docker

GitHub Actions

kubernetes

Advantages of YAML

- Lightweight
- It is **straightforward** to represent **complex mapping**
- Human friendly **readable** and **writable**
- Simple to **modify** with any **text editor**
- Suitable for **configuration settings**
- Support for **major programming languages**

What is YAML?

YAML is a data serialization Language

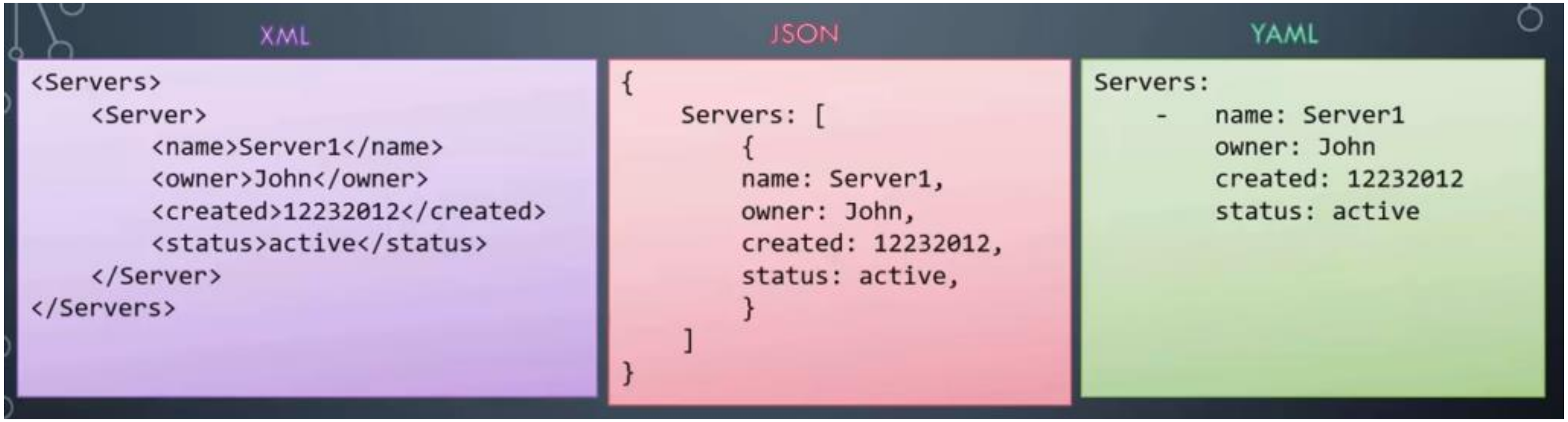
YAML stands for Ain’t Markup Language

Important Points

- It is case sensitive
- file extension is .yaml or .yml
- **Tabs are not** allowed,
- Some editors **allow spaces**

YAML format compared to other

- YAML is designed to be human -friendly and works well with other programming languages for everyday tasks.



XML

```
<Servers>
    <Server>
        <name>Server1</name>
        <owner>John</owner>
        <created>12232012</created>
        <status>active</status>
    </Server>
</Servers>
```

JSON

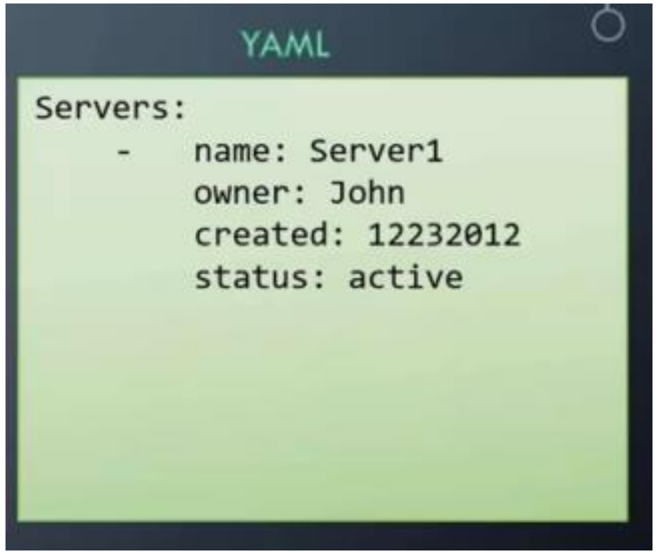
```
{
    Servers: [
        {
        name: Server1,
        owner: John,
        created: 12232012,
        status: active,
        }
    ]
}
```

YAML

```
Servers:
    -   name: Server1
        owner: John
        created: 12232012
        status: active
```

YAML format compared to other

- YAML files are used to store text data with arranged in hierarchical structure.



```
Servers:
    -   name: Server1
        owner: John
        created: 12232012
        status: active
```

Line Separation

Indentation

Syntax of YAML

- It supports Scalar types (Integer,strings,float,Boolean) as well as as collection types non-scalar (array,list).
- Comments
- Key-value pairs
- Objects
- Lists
- Booleans
- Multi-line strings
- Variables

YAML Syntax

- In YAML
  - **Indentation** represents the structure.
  - **Dashes (-)** are used to represent the lists and
  - **Colons(:)** are used to represent the key-value pair.

Example 1:

```yaml
datacenter:
  location: canada
  cab: 15
```

Example 2:

```yaml
animals:
  - dog
  - cat
  - mouse
```

Example 3:

```yaml
host: phl-42
```

세종대학교 SEJONG UNIVERSITY

YAML comments

- Comments are used to describe meaningful message about line of code.
- Usually in any technology, There are two types of comments
  - Inline or single line comments
  - Block level comments
- YAML has the syntax support for inline comments , but not for block level comments
- Inline comments are comments declared at the line of code Hashtag symbol is used at the start of the line to tell processor that it is comment

YAML Data types

- YAML has three types of data types:
  - Scalar
  - List
  - Dictionary

Scalar data type:

- The value of the scalar can be integer, float, Boolean, and string.
- Scalar data types are classified into two data types:
  - Numeric Data type
  - String

Numeric Data type

- There are three types of numeric data type:
  - Integer
  - Floating point numbers
  - Booleans
- An Integer data type can be decimal, octal, or hexadecimal.

Example 1:

```
age: 12345
octalexample: 012345
hexaexample: 0x12d4
```

Example 2:

```
boolenval1: True
booleanval2: False
fan: On
light: Off
```

String

- YAML strings are Unicode. In the following example, we are going to define a simple string, without using quotes.
- During the YAML file, we can set the value of a data variable to be null. Later, we can write a program to change the value of null to any other value.
- In YAML, we can write a multi-line string in a single line using > symbol. In this, a newline character(\n) will be ignored.

Example

```
str1: "the cost is 390\n"
str2: the cost is 390\n
```

Example

```
str1: null
str2: ~
```

Example

```
str: >
      this is a multi-line string it
      spans more than one
      line


str: |
     this is a multi-line string it
     spans more than one
     line
```

SEJONG UNIVERSITY

Lists

- We can define the list in a single line as follows:

Example

```
items: [6, 7, 8, 9, 10]
name: [six, seven, eight, nine, ten]
```

Example

```
items:
  - 6
  - 7
  - 8
name:
  - "six"
  - "seven"
  - "eight"
  - "nine"
```

Dictionaries

- If we want to write a complex YAML file which holds the complex data structure, we will use dictionaries.
- It is a collection of key: value pairs and each of the key: value pairs can be nested with a lot of options.

Example

```yaml
student1: "john"
hobbies:
  - music
  - reading
  - dancing
```

Example

```yaml
student2:
  fatherName: "William"
  motherName: "Marry"
  subjectDetails:
    subject1: 70
    subject2: 100
```

Example

```yaml
student: john
details:
  fatherName: william
  motherName: marry
more:
    - subject1
    - subject2
```

YAML Anchors

- It allows us to store and reuse data within our YAML file.
- When we enter data into our YAML file, we might find that some data or an entire collection of data gets reused throughout the file, as shown below:

```yaml
definitions:
  steps:
    - step: &build-test
        name: Build and test
        script:
          - mvn package
        artifacts:
          - target

pipelines:
  branches:
    develop:
      - step: *build-test
    main:
      - step: *build-test
```

Parse YAML

- pip install pyyaml

```
import yaml

with open('config.yml', 'r') as file:
    prime_service = yaml.safe_load(file)

print(prime_service['prime_numbers'][0])
print(prime_service['rest']['url'])
```

config.yml

```
rest:
  url: "https://example.org/primenumbers/v1"
  port: 8443
prime_numbers: [2, 3, 5, 7, 11, 13, 17, 19]
```

What an Open-Source Project Needs

1

Version control and Bug tracking

Enables developers to manage code changes conveniently, including reverting and "change porting". Enables everyone to watch what's happening to the code. Enables developers to keep track of what they're working on, coordinate with each other, and plan releases.

2

Website

Primarily a centralized, one-way conduit of information from the project out to the public. The web site may also serve as an administrative interface for other project tools

3

Documentation

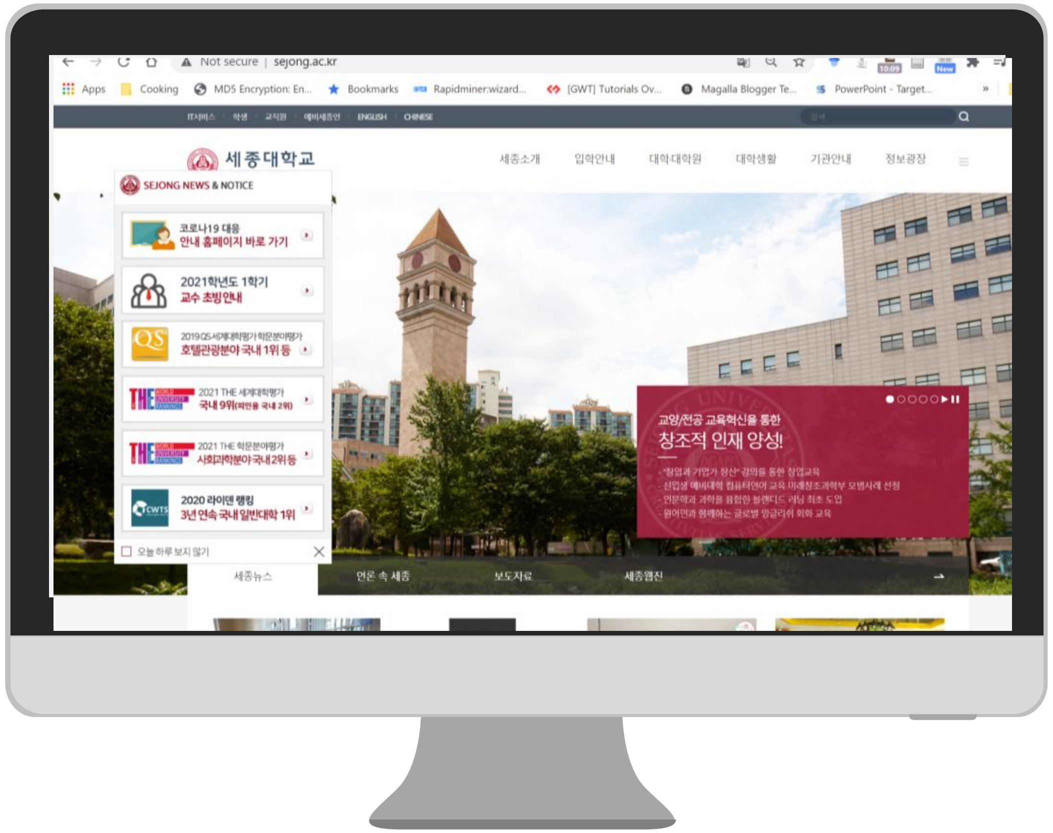
You can simply impress your audience and add a unique zing and appeal to your Presentations.

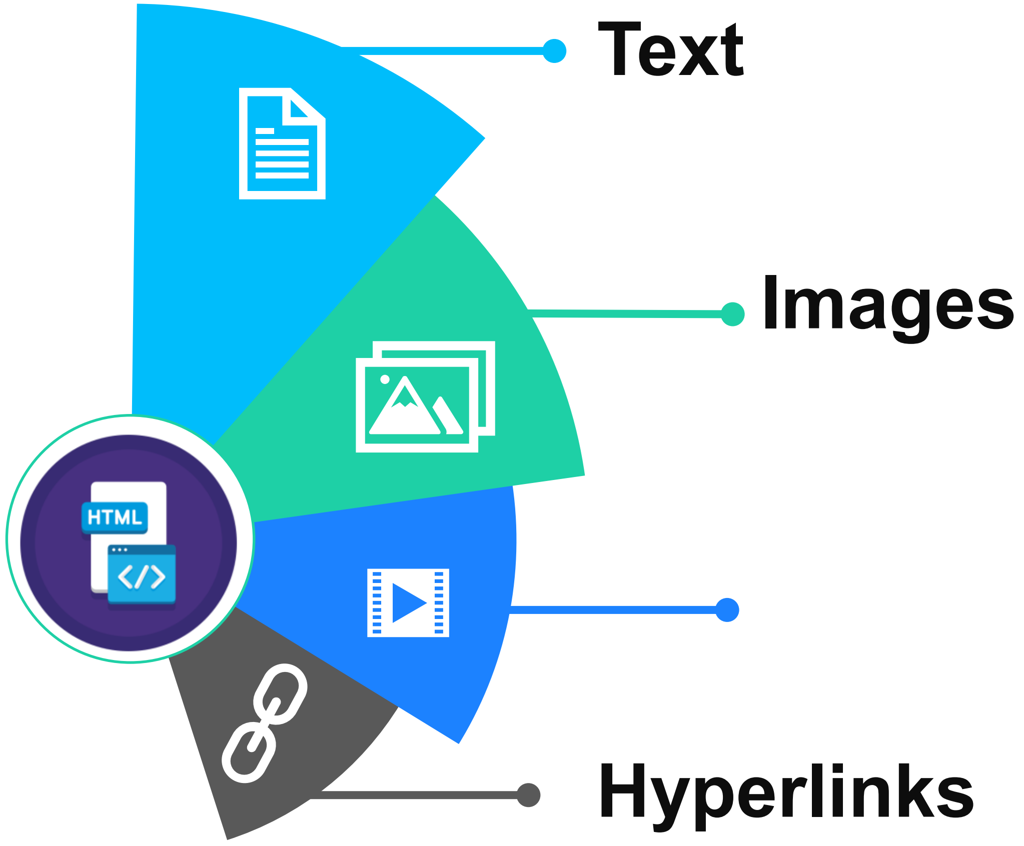
4

Mailing lists / Message forums

Usually, the most active communications forum in the project, and the "medium of record. "A place for quick, lightweight discussions and question/answer exchanges. Not always archived completely

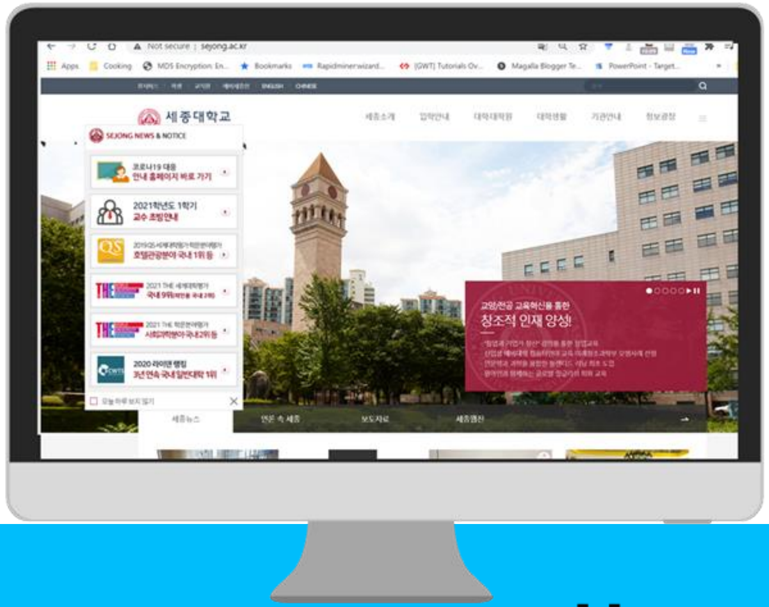
What is a Website?





HTML Document

What is a Website?



Domain Name

Collection of Web pages

Interactive & Static content

Navigations

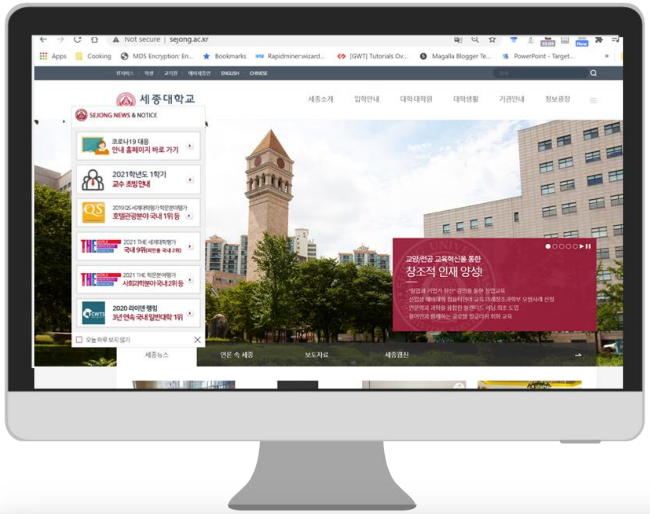
Hyperlinks

Clickable

Interactive

Location/Action

What is a Website?

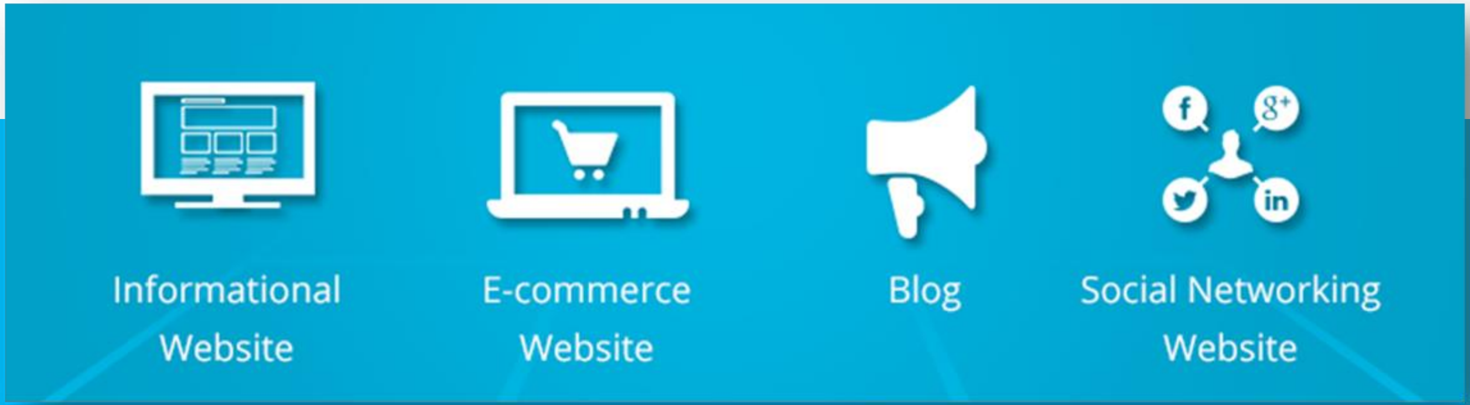


Domain Name

Collection of Web pages

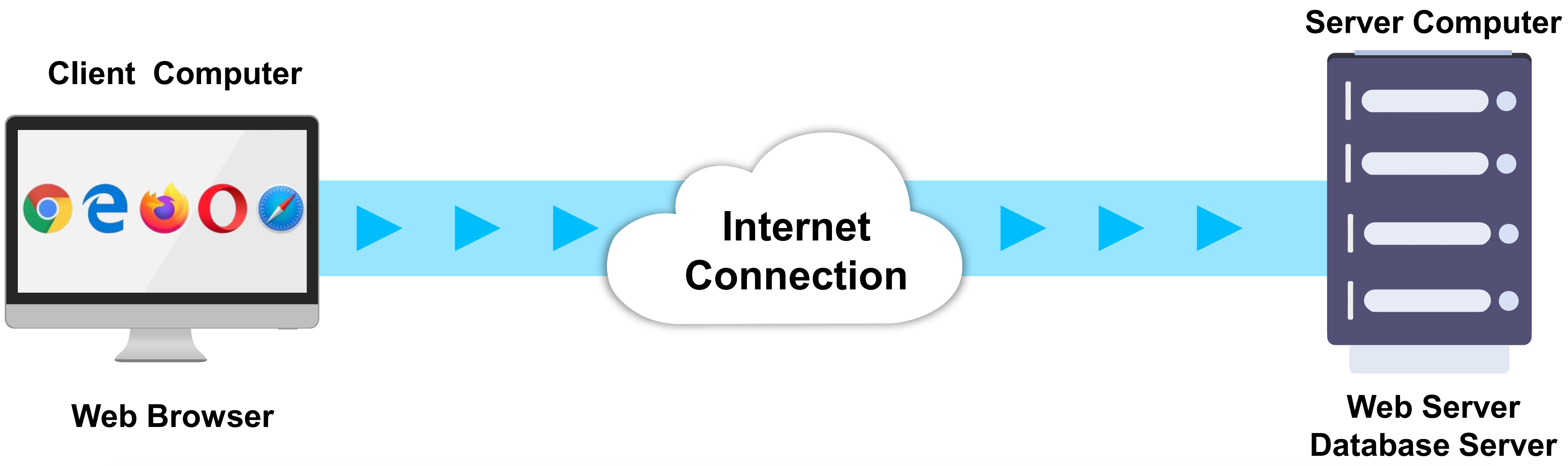
Interactive & Static content

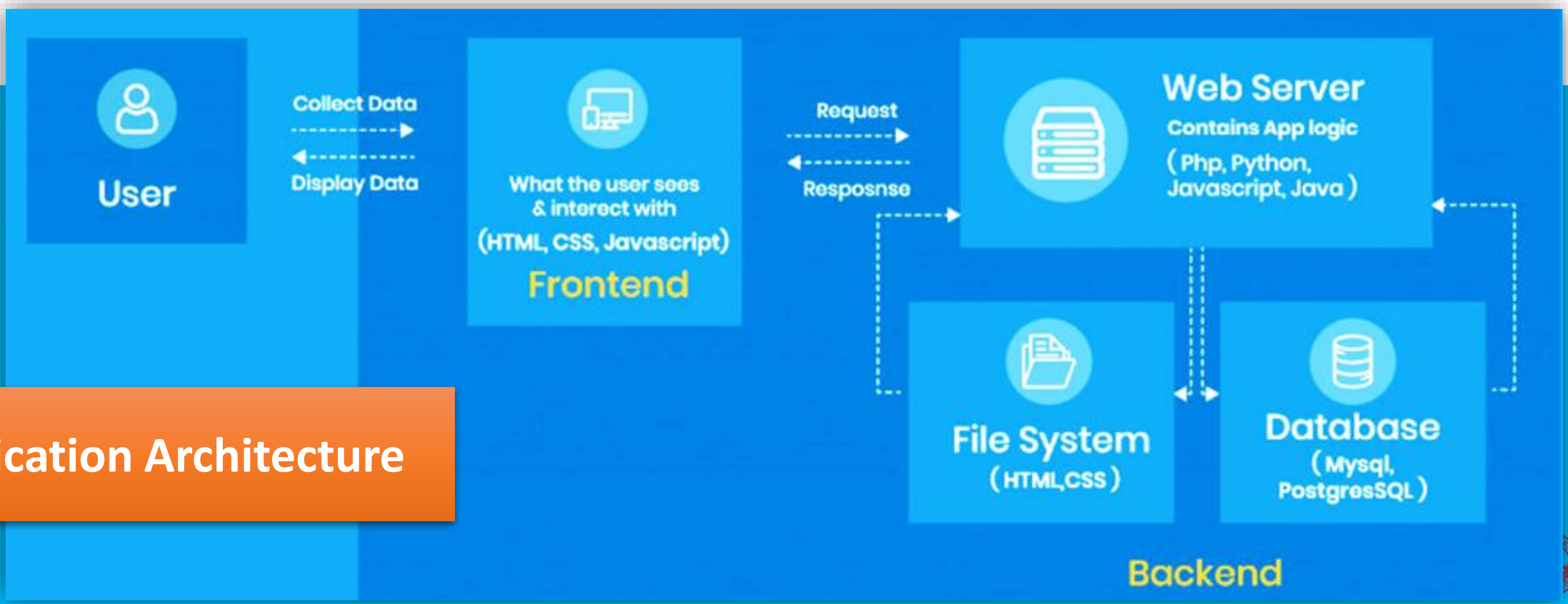
Navigations



TYPE OF WEBSITES

Accessing a web site





Web Application Architecture

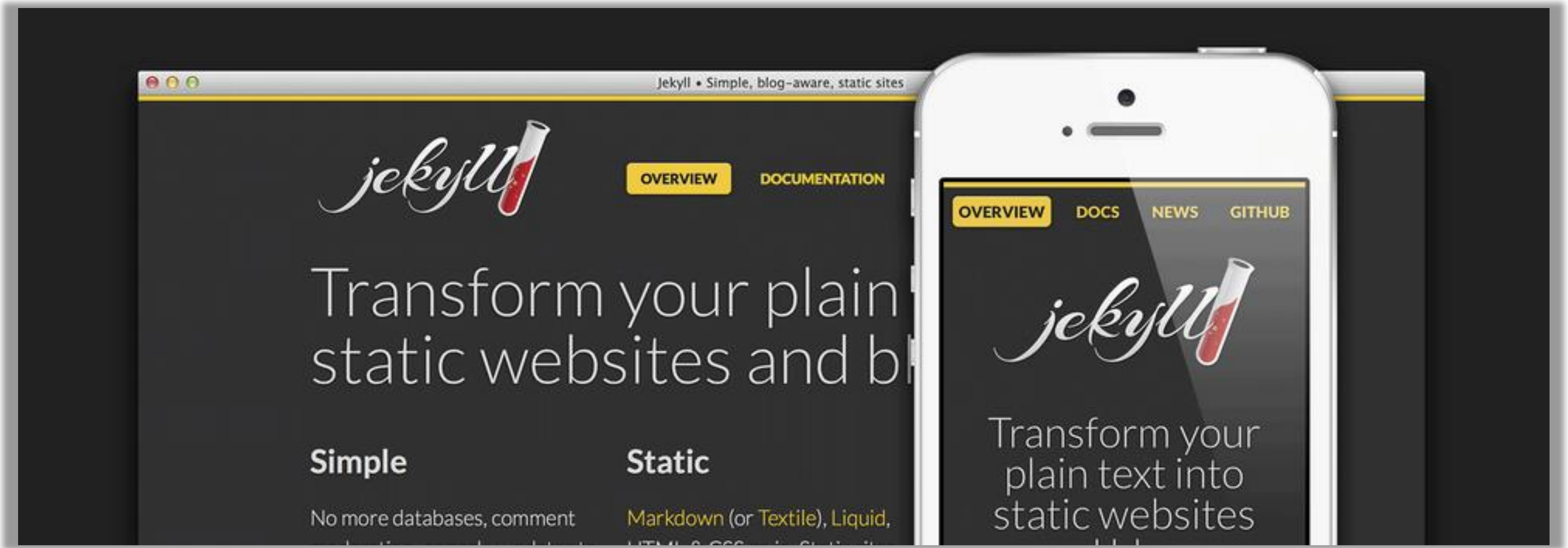
WEB DEVELOPMENT

Web development is the work involved in developing a Web site for the Internet or an intranet. Web development can range from developing a simple single static page of plain text to complex Web-based Internet applications, electronic businesses, and social network services.

Wikipedia

Jekyll

Transform your plain text into static websites and blogs.



QuickStart with Jekyll

- Jekyll is a static site generator.
- It takes text written in your favorite markup language and uses layouts to create a static website.
- You can tweak the site’s look and feel, URLs, the data displayed on the page, and more.

Prerequisites

- Jekyll requires the following:
  - Ruby version 2.5.0 or higher
  - RubyGems
  - GCC and Make
- Jekyll is a Ruby Gem that can be installed on most systems.

Instructions

1. Install all prerequisites.
2. Install the jekyll and bundler gems.

```
gem install jekyll bundler
```

3. Create a new Jekyll site at ./myblog.

```
jekyll new myblog
```

4. Change into your new directory.

```
cd myblog
```

5. Build the site and make it available on a local server.

```
bundle exec jekyll serve
```

6. Browse to http://localhost:4000

Basic Jekyll structure

- jekyll is a “convention over configuration” framework, and the basic setup is quite simple.
- You should now see two main folders - **_posts** and _site (you can ignore .jekyll-cache altogether):
- **_posts** is simply where your blog posts will go in future.
- **_site** is where your finished site is “built”. In other words, this is where your viewable website will be created - ready for deployment. Any files such as CSS, JS, and images will also end up here in an assets folder.

Basic Jekyll structure

- There are some other files worth mentioning too:
  - **_config.yml** is where you manage the configuration for your project, like global variables, “collections”, or default names/paths. This where a lot of customization is done.
  - **.gitignore** is for files/folders you don’t want to save into version control (e.g., information you don’t want available publicly).
  - **Gemfile/Gemfile.lock** is how you manage any extensions to Jekyll.

- You might have also noticed a number of Markdown files. These are used to make it easy to write content - especially blog posts.
- In fact, there are no about.html and index.html by default. These are actually built into HTML - automatically - from about.markdown and index.markdown into the _site folder.

Further conventions

- In addition to these basics, you can also manually add other folders with specific names that Jekyll will recognize.
- **_data:** any “data” files your site needs (like a small database)
- **_drafts:** posts that you don’t want automatically published
- **_layouts:** files that you want to use as a “frame” for specific pages - e.g., the layout for all of your posts
- **_includes:** parts of HTML files you want to reuse in multiple pages - e.g., navigation, footer.

What is front matter?

- Front matter is an area at the top of your HTML/Markdown documents that lets you write variables and even content for your pages.
- It uses **YAML**, a simple and friendly serialization language

- To write a simple YAML variable, use **key: value** notation, with a colon.

Posts

- Blogging is baked into Jekyll. You write blog posts as text files and Jekyll provides everything you need to turn it into a blog
- The **_posts** folder is where your blog posts live. You typically write posts in **Markdown**, **HTML** is also supported.
- To create a post, add a file to your **_posts** directory with the following format:

```
YEAR-MONTH-DAY-title.MARKUP
```

- All blog post files must begin with **front matter** which is typically used to set a layout or other meta data. For a simple example this can just be empty:

Posts - example

```
---
layout: post
title:  "Welcome to Jekyll!"
---

# Welcome

**Hello world**, this is my first Jekyll blog post.

I hope you like it!
```

Post- Drafts

- Drafts are posts without a date in the filename.
- They're posts you're still working on and don't want to publish yet.
- To get up and running with drafts, create a **_drafts** folder in your site's root and create your first draft

- To preview your site with drafts,
- run **jekyll serve** or **jekyll build** with the --drafts switch.
- Each will be assigned the value modification time of the draft file for its date, and thus you will see currently edited drafts as the latest posts.

Posts- Tags and Categories

Tags

- Jekyll has first class support for tags and categories in blog posts..
- Tags for a post are defined in the post’s front matter using either the key tag for a single entry or tags for multiple entries.

```
tag: classic hollywood
```

"classic hollywood"

```
tags: classic hollywood
```

["classic", "hollywood"]

Posts- Tags and Categories

Categories

- Categories of a post work similar to the tags:
- They can be defined via the front matter using keys category or categories (that follow the same logic as for tags)
- All categories registered in the site are exposed to Liquid templates via site.categories which can be iterated over.

```
category: classic hollywood
```

```
categories: classic hollywood,
```

세종대학교
SEJONG UNIVERSITY

Pages

- Pages are the most basic building block for content.
- They’re useful for standalone content (content which is not date based or is not a group of content such as staff members or recipes).
- The simplest way of adding a page is to add an HTML file in the root directory with a suitable filename.
- You can also write a page in Markdown using a .md extension which converts to HTML on build.
- If you have a lot of pages, you can organize them into subfolders

Permalinks

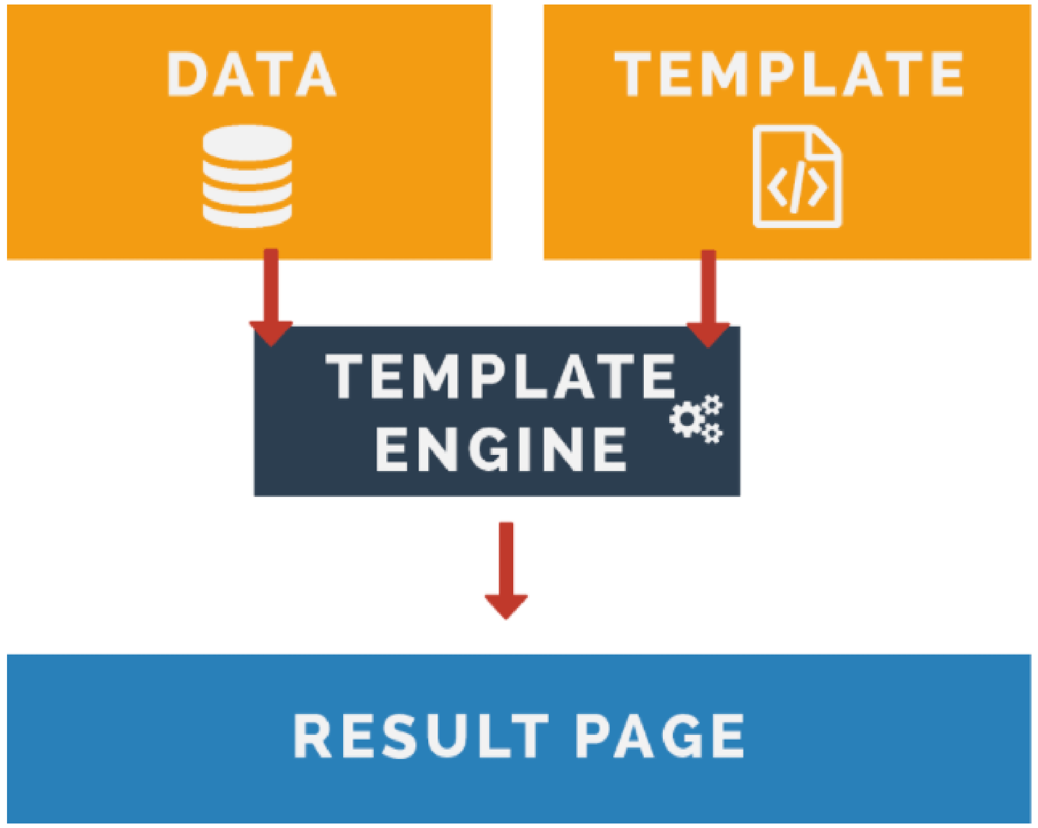
- Permalinks are the output path for your pages, posts, or collections.
- They allow you to structure the directories of your source code different from the directories in your output.
- The simplest way to set a permalink is using front matter.
  - You set the permalink variable in front matter to the output path you'd like.

```
---
permalink: /about/
---
```

```
---
permalink: /:Categories/:year/:month/:day/:title.html
---
```

What is Liquid?

- Liquid is a **templating language** used in Jekyll to **process your site’s pages**.
- In other words, it helps you make your HTML pages a bit more **dynamic**, for example adding logic or using content from elsewhere.
- This doesn’t require any setup - we can just start using it.



Liquid basics

- To use Liquid in your pages, you first need pages to have “front matter” notation (our next lesson - don’t worry for now) at the top of our page:

```
---
---
<!DOCTYPE html>
```

- Offical Site of Liquid
  - https://shopify.github.io/liquid/

Liquid basics

- For Liquid code itself, there are two types of tags:
- For outputting content into a page, use two curly brackets on each side:

```
{{ content }}
```

- For logic/code, use a curly brackets and % sign on each side:

```
{% if condition == true %}
```

- Importantly, logic/code blocks must also have an “end” statement, for example {% endif %}, {% endfor %}

Liquid basics

Filters

- A filter is something that you can use to change strings (text) or manipulate arrays (lists of items).
- To use a filter, separate the content you want to filter with a | sign and use a filter keyword.
- Filters can also be chained and take “arguments” - that is, extras to modify output more specifically.

- There are many, many filters available that you might want to learn about , but here we’ll just look at some common examples and output:

Liquid basics

Filters

- All uppercase
  - {{ "uppercase" | upcase }} = UPPERCASE
- All lowercase
  - {{ "LOWERCASE" | downcase }} = lowercase
- Length of a string
  - {{ "How long am I?" | size }} = 14
- {{ "Copyright " | append: "My Blog" }} = Copyright My Blog
- Simple date formatting - international format

- {{ "2021-01-01T00:00:00Z" | date_to_long_string }} = 01 January 2021

Liquid basics

Variable

- Variable tags create new Liquid variables using **assign** keyword

```
{% assign my_variable = false %}

{{my_variable}}
```

- Captures the string inside of the opening and closing tags and assigns it to a variable. Variables created using capture are stored as strings.

```
{% capture my_variable %}I am being captured.{% endcapture %}
{{ my_variable }}
```

Liquid basics

Variable – Jekyll

- Jekyll traverses your site looking for files to process. Any files with front matter are subject to processing.
- For each of these files, Jekyll makes a variety of data available via Liquid. The following is a reference of the available data.

https://jekyllrb.com/docs/variables/

Liquid basics

Tags

- A condition is a great way to display content on your page based on decisions.
- These also combine well with “logical operators” for making comparisons:

Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
and	Both condition A and B
or	Either condition A or B

Liquid basics

Tags - Control flow

- The most common type of condition in Liquid is the "if" statement. Here's an example of displaying a title depending on a title variable:

```
{% assign title = "home" %}
{% if title == "home" %}
  <h1>Welcome to my homepage!</h1>
{% endif %}
```

```
{% assign title = "home" %}
{% if title == "home" %}
  <h1>This is the homepage</h1>
{% elsif title == "about" %}
  <h1>This is the about page</h1>
{% else %}
  <h1>Welcome!</h1>
{% endif %}
```

Liquid basics

Tags – Loops

- A loop makes it easier to work with arrays.
- With a loop, we can make this easier and more dynamic. The syntax for a loop is for <variable> in <list of items>, where “variable” can be anything you choose:

```
{% assign products = "Kiwi,Tui,Kea,Karariki,Weka" | split: "," %}
<ul>
  {% for item in products %}
    <li>{{ item }}</li>
  {% endfor %}
</ul>
```

WJ
Jekyll collection

Collections

- Collections in Jekyll are quite similar to the posts that we created in the previous lesson. So what’s the difference? Here’s a simple summary:
  - Use posts when you want to write independent articles, with a publishing date.
  - Use collections when you want to group related content, which can have its own page, but date is unimportant.
- Collections are a great way to group related content like members of a team or talks at a conference.

Collections

Setting

- To use a Collection you first need to define it in your _config.yml. For example here's a collection of staff members:

```
collections:
  - staff_members
```

- Create a corresponding folder (e.g. <source>/_staff_members) and add documents.
- Manually Ordering Documents
  - You can also manually order the documents by setting an order metadata with the filenames listed in the desired order. For example, a collection of tutorials would be configured as:

```
collections:
  staff_members:
    order:
      - jamil.md
      - ali.md
```

Collections

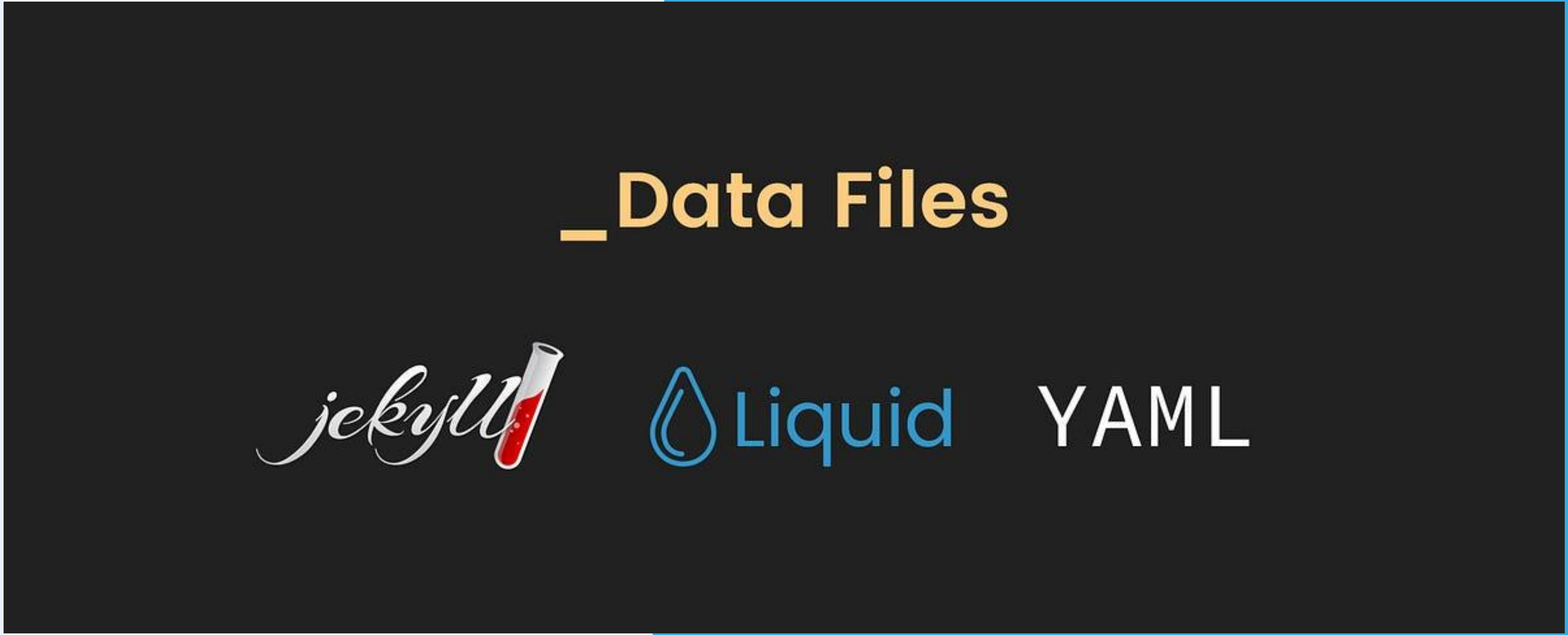
Adding content

- The filename is ./_staff_members/jami.md with the following content:

```
---
name: Jamil
position: Developer
---
Jane has worked on Jekyll for the past *five years*.
```

- Now you can iterate over site.staff_members on a page and output the content for each staff member.

```
{% for staff_member in site.staff_members %}
  <h2>{{ staff_member.name }} - {{ staff_member.position }}</h2>
  <p>{{ staff_member.content | markdownify }}</p>
{% endfor %}
```


_Data Files
jekyll
Liquid
YAML

Data files

- Sometimes you also need supplemental data to use on pages, but not as its own page, much like from a database or API.
- Jekyll allows you to create data files and access them globally
- This way you can maintain your own mini-databases, but with very little setup needs.
- A number of file formats are supported, including JSON, YAML, CSV (comma-separated values), and TSV (tab-separated values) in the _data directory.
  - Note that CSV and TSV files must contain a header row

Data files

Data Folder

- The _data folder is where you can store additional data for Jekyll to use when generating your site.
- These files must be YAML, JSON, TSV or CSV files (using either the .yml, .yaml, .json, .tsv, or .csv extension), and they will be accessible via site.data.

Data files

Example

- Here is a basic example of using Data Files to avoid copy-pasting large chunks of code in your Jekyll templates:

_data/members.yml

```yaml
- name: Eric Mill
  github: konklone

- name: Parker Moore
  github: parkr

- name: Liu Fengyun
  github: liufengyun
```

_data/members.csv

```
name,github
Eric Mill,konklone
Parker Moore,parkr
Liu Fengyun,liufengyun
```

Data files

Example

- This data can be accessed via site.data.members

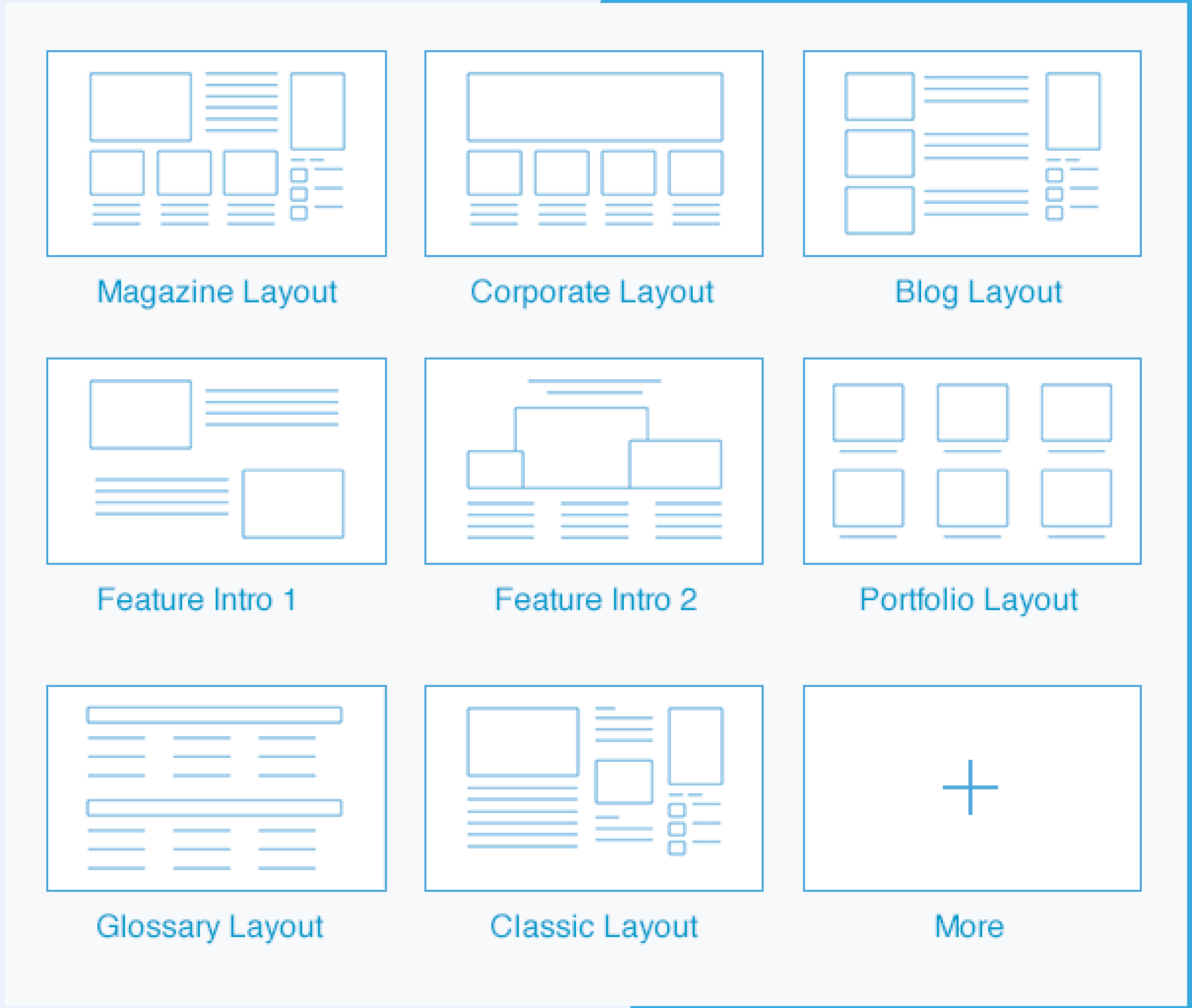
_data/members.yml

```yaml
- name: Eric Mill
  github: konklone

- name: Parker Moore
  github: parkr

- name: Liu Fengyun
  github: liufengyun
```

```html
<ul>
{% for member in site.data.members %}
  <li>
    <a href="https://github.com/{{ member.github }}">
      {{ member.name }}
    </a>
  </li>
{% endfor %}
</ul>
```


Magazine Layout
Corporate Layout
Blog Layout
Feature Intro 1
Feature Intro 2
Portfolio Layout
Glossary Layout
Classic Layout
More

layouts

- When writing a website in HTML, you will probably notice that many sections stay the same across multiple pages, such as head, footers, and navigation.
- If your site contains more than a few pages, that’s a lot of content to copy and paste - and any changes need to be made across all pages.
- Jekyll gives us an easy solution to this problem - layouts.
- Layouts are templates that wrap around your content.
- They allow you to have the source code for your template in one place so you don’t have to repeat things like your navigation and footer on every page.

- Layouts live in the _layouts directory.
- The convention is to have a base template called default.html and have other layouts inherit from this as needed.

layouts

- The first step is to put the template source code in default.html. content is a special variable, the value is the rendered content of the post or page being wrapped.
- You can access the page variables and font-matters in template layout

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>{{ page.title }}</title>
    <link rel="stylesheet" href="/css/style.css">
  </head>
  <body>
    <nav>
      <a href="/">Home</a>
      <a href="/blog/">Blog</a>
    </nav>
    <h1>{{ page.title }}</h1>
    <section>
      {{ content }}
    </section>
    <footer>
      &copy; to me
    </footer>
  </body>
</html>
```

layouts

Inheritance

- Layout inheritance is useful when you want to add something to an existing layout for a portion of documents on your site.
- A common example of this is blog posts, you might want a post to display the date and author but otherwise be identical to your base layout.

```
---
layout: post
---
<p>{{ page.date }} - Written by {{ page.author }}</p>

{{ content }}
```

Includes

- Sometimes we have smaller page fragments that we want to remain consistent over multiple pages.
- Great examples of header and footer.
- Jekyll includes allow you to break down your pages into smaller “components” like navigation, section titles, and footers - there are many potential use cases.
- The include tag allows you to include the content from another file stored in the _includes folder:

```
{% include footer.html %}
```

Includes

- Setting up our includes is much like layouts - we need to create an _includes folder for Jekyll to recognize, and then we can put our HTML fragments in it.
- For our first example, let’s take our existing navigation and footer from our default.html layout and place them in their own files:
  - Create _includes/nav.html. Cut and paste all of the <header> element from layouts/default.html into this file.
  - Create _includes/footer.html. Cut and paste all of the <footer> element from layouts/default.html into this file.
- Lastly, to use our new includes, simply add these in the place of the content we have moved:

Includes

```
---
---
<!DOCTYPE html>
<html lang="en">
 ... rest of head ...
<body class="page">
  {% include nav.html %}
    <main class="main-content">
      <div class="container">
        <div class="page-content">
          {{ content }}
        </div>
    </main>
    {% include footer.html %}
</body>
</html>
```

Includes

pass parameters to includes

- It’s great being able to break our site down further, but what if we want to create a component that changes as we need it, like a number of social media posts?
- Normally, we would just have to copy and paste embed code from a provider, but Jekyll offers another solution: parameters.
- Let’s create a YouTube component that we can put on our page. Create youtube.html in your _includes folder and paste this code into it:

Includes

pass parameters to includes

```
<div class="spacing youtube">
<iframe width="560" height="315"
    src="https://www.youtube.com/embed/{{ include.youtube_id }}"
    frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media;
gyroscope; picture-in-picture"
    allowfullscreen>
</iframe>
</div>
```

```
<p class="featured">Featured posts</p>
<h2 class="heading-secondary dark-blue">Latest videos</h2>
<div class="includes-grid">
  {% include youtube.html youtube_id="7W7hEUGtv4U" %}
  {% include youtube.html youtube_id="E3a88_SjJR0" %}
</div>
```

GitHub Pages

- GitHub enable to websites for you and your projects.
- Hosted directly from your GitHub repository. Just edit, push, and your changes are live.

GitHub Pages

- There are no databases to set up
- No servers to configure in many cases
- You don't even have to know HTML everything just works

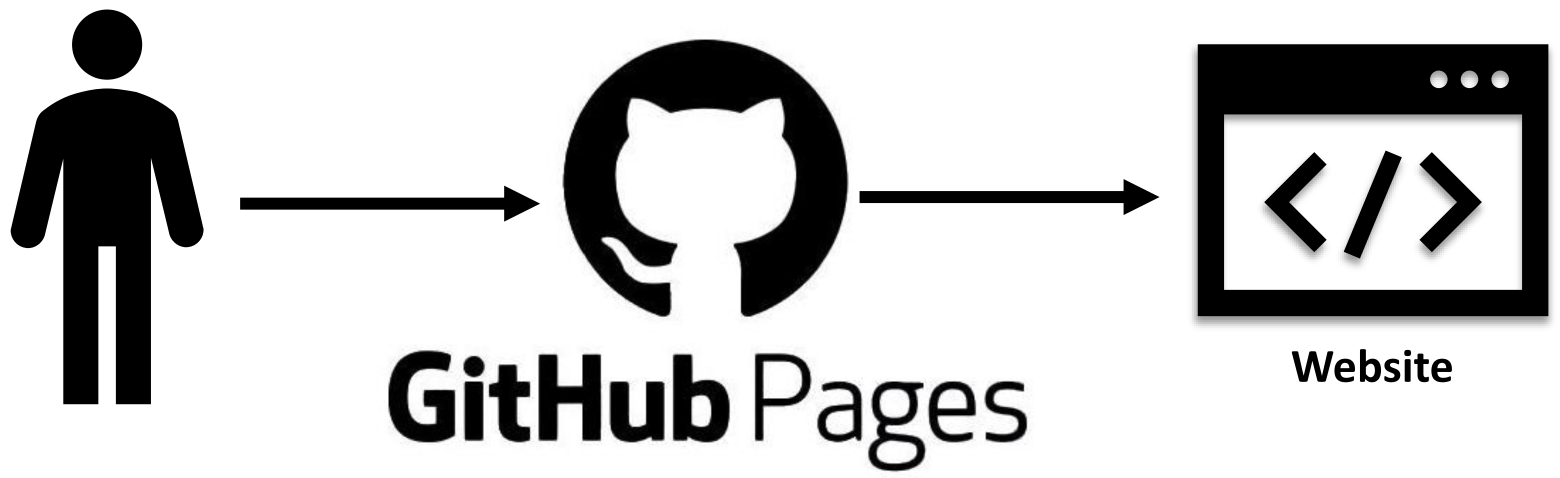
No databases

No need of servers

HTML

GitHub Pages

- If you already use the GitHub, then it is the most direct path to build website



Reading Materials

- https://www.w3schools.io/file/yaml-introduction/
- https://www.tutorialspoint.com/yaml/index.htm
- https://www.javatpoint.com/yaml

- https://jekyllrb.com/resources/
- https://jekyllrb.com/tutorials/video-walkthroughs/
- https://medium.com/blueeast/jekyll-and-data-files-with-real-time-example-6ea704213111

Thanks

Office Time: Monday-Friday (1000 - 1800)

You can send me an email for meeting, or any sort of discussion related to class matters.

jamil@sejong.ac.kr

세종대학교
SEJONG UNIVERSITY