Programming Standards

Center for New Testament Restoration

By Alan Bunning

September 17, 2025

Introduction

All the programming code used by the Center for New Testament Restoration (CNTR) is meant to adhere to the following programming standards. Such standards have the benefit of reducing errors, improving efficiency, providing consistency, enhancing readability, and increasing maintainability. Theses standards are both descriptive and prescriptive – descriptive in the sense that these *were* the practices and justifications for the code that was written (whether for the better or worse), and prescriptive in the sense that they should then be maintained to keep a uniform style until change is warranted. These standards may evolve over time as new versions of the languages are released, industry conventions evolve, or the preferences of new team members are taken into consideration. Consequently, some programs at any given time may be found to be out of compliance, but the intent is that they should be modified to adhere to the current standards whenever programs are updated.

These standards deviate slightly from other similar documents in a few cases with the goal of of improving portability to any future platform by using constructs in a more generic manner that would be more similar to other programming languages. This provides the additional benefits of reducing the learning curve for programmers who are new to this environment, and making the code more familiar to experienced programmers coming from other backgrounds.

Many heated arguments have occurred over various programming standards and some guides merely state what appear to be arbitrary preferences. Each guideline stated here, however, is provided with its justification so that they may be reconsidered when the given rationale no longer applies. Other programming precepts that are not addressed in this document are left to the programmer's discretion and new standards for those areas should be proposed when applicable.

JavaScript

General

- All programs should be written in pure JavaScript in compliance with the most recent ECMAScript version (currently ECMA-262).

  [*The JavaScript language was chosen for its inherent longevity as it serves as the foundation of website development, and will continue to do so for a very long time. JavaScript remains the most popular language by sheer volume and has a very substantial support base. It is free and is easy to run in any browser without any additional software. JavaScript provides a single one-stop solution as it can be used for both client-side components, and for server-side components using Node.js. Since JavaScript has to be used anyway for web page development, the ability to use it for all software development means there is only one language to be mastered. JavaScript is not by any means a perfect language, but it is certainly more than adequate for the task and is being improved in subsequent editions.*]

- No additional frameworks built on JavaScript should be used.

  [*Such frameworks are technically different languages, their longevity has historically been short-lived, they carry a good amount of additional overhead, there is a steep learning curve in mastering their complexities, and this project is currently not complicated enough to realize significant benefits from using them.*]

- The major program sections should be arranged in the following order:

  1. included modules
  2. global declarations
  3. main program
  4. other functions and closures

  [*The included modules and global declarations appear first because they apply throughout the program. Global declarations include things related to the system environment (i.e. input/output) that are referenced in multiple functions. The main program is the first function for it shows the high-level logic of the whole process. Placing it as near the top as possible avoids unnecessary scrolling to understand the program's purpose (compared to placing it at the bottom).*]

- ECMAScript Modules (ESM) should be used using `import` and `export` statements and all they should all use the `.js` file extension.

  [*The use of ESM modules is the native convention used by browsers and is supported by Node.js, so other non-standard module systems such as CJS should not be used. Only having to deal with one module system greatly improves interoperability and the reuse of code.*]

Naming Conventions

- Variable and function names should be in camelCase format with a lowercase first word and subsequent words beginning with an uppercase letter. Acronyms should also follow this rule instead of being all uppercase. For example:

  ```
  let numBlackWidgits, httpString, websiteUrl;
  ```

  [*This allows multiple words to be distinguishable without having to add extra characters between them.*]

- Names of arrays and objects used as associative arrays should be plural in concept (often ending with an "s"), while other object names should be singular.

  [*Arrays and object names being used as associative arrays contain multiple entries so they are plural. Other object names should be singular even though they have multiple properties.*]

Functions

- The main program should be written as an Immediately Invoked Function Expression (IIFE). For example:

  ```
  void function main() {
    ...
    }();
  ```

  [*This allows the main entry point of the program to be named (similar to a C program) and allows its scope to be encapsulated; otherwise, its variables would be global. It also allows asynchronous operations to be performed in it by adding the* `async` *keyword.*]

- It is suggested that the initialization of data structures be placed in an `init()` function that comes after the main program, but exists within the scope of the main program.

  [*This makes the data structures available to the main program without cluttering the main program and without making them global variables.*]

- Functions should normally be defined using the traditional `function` keyword instead of the newer arrow notation. The arrow notation, however, may be used in methods and functions to simplify readability, particularly with bodies consisting of one line of code.

  [*This improves the general readability of the code for those coming from other programming languages, without sacrificing any efficiency. Also note that arrow functions are also not hoisted.*]

- Closures should be written as traditional functions and named beginning with the underscore character. They should be assigned as constants in the global declarations area with the same name minus the underscore character. For example:

  ```
  const functionName = _functionName();
  ```

[*Closures must be declared before they can used, so this allows them to be called anywhere in the program, while the code for them can be written below the main function along with the other functions.*]

- Input and output should be handled synchronously whenever possible, using the `async` and `await` statements as necessary.

  [*This is the way most other computer languages operate, making it easier for programmers coming from other platforms to understand. Asynchronous programming is sometimes the appropriate solution for some applications and can greatly improve performance, but it should be avoided when possible as it is harder to debug and test, and often results in "callback hell" which makes code more difficult to read and maintain.*]

Variables

- All variables should be declared as near to the place they are used for first time, often just before a scope or at the top of a scope.

  [*This is considered to be superior to the legacy practice of declaring all variables at the top of a function because it introduces the variables in the area where they are relevant, and eliminates much scrolling back and forth to declare variables. The compiler still prevents variables from being declared more than once, so that is not a concern.*]

- All variables should normally be declared with either the `let` or `const` statement.An exception is made for the `var` statement, which may be used for the control variable of a `for` loop if its value is needed after the loop ends.

  [*The* `let` *and* `const` *statements are limited to the local scope they reside in. The scope of the* `var` *statement, however, applies to the whole function they reside in, which some find to be confusing, so it should be used with caution.* `const` *is normally used to declare arrays and objects to prevent the whole array or object from being reassigned, but still allows the elements and properties to be modified.*]

- The use of global variables should generally be avoided. If they are used, they should be declared in the global declarations area.

  [*The use of global variables usually indicates a bad design that could be restructured to only pass the variables necessary for function. Of course, there are always exceptions where they may make the code more compact and easier to understand.*]

- Arrays and objects should be declared showing their types. For example:

  ```
  let words = [], matrix = {};
  ```

  [*Although both are object types in JavaScript, it makes the difference between an ordered collection (array) and key-value pairs (object) clear when they are defined.*]

Statements

- There is flexibility in the choice of loop constructs, but there are some general recommendations. The `for of` statement should be preferred for loops with multi-line bodies. The `for` statement should be used when the index variable needs to be referenced or altered. The `forEach` method is generally used with single-line bodies.

  [*The* `for of` *loop is easier to read, allows the use of statements like* `continue`*,* `break`*, and* `await`*, iterates over other elements beside arrays, and is purportedly faster. Using* `forEach` *with single-line bodies is acceptable when compactness is helpful, or for when the index variable is also needed.* `forEach` *has known problems involving input/output and should not be used in those cases.*]

- The methods that involve looping should generally use the default `(e, i, a)` representing the element, index, and array, but any of these variables may be renamed for greater clarity.

  [*The* `i` *variable is already normally used for indexes in loops, so the adding the others defaults merely provides some consistency.*]

- The `slice` function should always be used instead of `substring` or the deprecated `substr`.

  [*The* `slice` *function has superior functionality by allowing negative numbers, and is purportedly faster.*]

- Error handling using `try` and `catch` should be used sparingly, and only where it meaningfully improves the program's behavior.

  [*Adding* `try` and `catch` *often only complicates the code and wastes time checking for conditions that should not happen. Most simple programs either work or they don't and if they don't the programmer should find out why and fix it. One valid reason to use them, however, is when an error would visibly effect the user's experience. This is particularly true when receiving external data out of the programmer's control. A user should never experience a system error message or abnormal program termination.*]

Formatting

- Program documentation should use a subset of the JSDoc 3 format by placing the appropriate tags in comments. Each file should include a block comment at the top that includes the tags `@file`, `@copyright`, and `@license`. Each exported function should include a block comment that includes the tags `@param` and `@returns` where applicable. Other functions do not require any tags, but should at least have a block comment describing its purpose. For example:

```
/**
 * program description
 */
```

  [*These JSDoc tags have become relatively standard and can be used to automatically generate program documentation. The entire JSDoc framework is not utilized and the built-in document*

generated is not particularly useful, but these minimal tags are useful allowing `grep` *and other programs to access them. The tags are needed for exported functions because they have an external context as stand-alone components designed to be used in other programs. But the tags are not needed for other internal functions because the context of the program explains their usage.*]

- Each programming construct level should be indented by two spaces.

  [*Many a religious war has been fought over the amount of indentation, but two spaces are the minimum needed for decent readability, so the minimum is adopted to save space.*]

- The curly braces should follow the Ratliff indentation style (also known as the banner style).

  ```
  construct {
    ...
    }
  ```

  [*This style is the most conducive to the generic concept used in most other languages, where all statements belonging to the construct are indented under the construct. Despite the popularity of the 1TBS (One True Brace Style) developed by Kernighan and Ritchie, the Ratliff style was designed to be objectively better for visual scanning, since the headers of any block are the only thing extended at each level. An argument could be made that the opening brace could be on a line by itself, but it is included with the construct to reduce the number of lines.*]

- The body of any construct that contains multiple lines of code should usually be encapsulated in curly braces.

  [Although the compiler allows almost every construct to use a one-line body without curly braces, nesting multiple constructs each with one-line bodies can cause confusion, especially among programmers who were falsely taught that each constructs always requires curly braces.]

- One space should occur after any comma, colon, or semicolon (if there are other items following it). But there should not be a space before any of them.

  [*This follows the conventions of the English language and helps improve the readability of items that occur in lists.*]

- There should be at least two spaces separating a programming statement from a same-line comment that follows.

  [*This helps improve its readability, for if there is only one space, or no space, it becomes too congested. Such comments may also be lined up with those above or below by adding additional spaces if desired.*]

- Double quotation marks should generally be used for strings.

  [*This has been the historical practice for most programming languages. JSON also requires double quotation marks which is more consistent with this practice. Single quotation marks came later are reserved for HTML/CSS attributes.*]

- Template literals should be used whenever possible instead of concatenating variables and text together with the “+” symbol in output strings.

  ```
  `text ${variable}`
  ```

  [*This makes it easier to visualize how the output appears, and usually takes up fewer characters.*]

JavaScript Reserved Words

```
await
break
case
catch
class
const
continue
debugger
default
delete
do
else
enum
export
extends
false
finally
for
function
if
implements
import
in
instanceof
interface
let
new
null
package
private
protected
public
return
super
switch
static
this
throw
try
true
typeof
var
void
while
with
yield
```

MySQL

General

- The latest version of MySQL server (currently 8.0.41) should be used for all database applications.

  [*MySQL has a large developer support base and wide compatibility with other platforms. It has a robust feature set and is scalable for large datasets. It is open source software and is free to use. PostgreSQL was also heavily considered, but had less developer support, was slower for basic operations, was not as compatible with other operating systems, and its advanced features would not be used anyway because that would make it less portable.*]

- Generic implementations of the latest SQL standard (currently ISO/IEC 9075:2023) should be adhered to whenever possible.

  [*This will make it more compatible with other SQL servers and easier to port should the need ever arise.*]

Design

- Databases should normally be designed in third normal form, but there are several situations where it is acceptable to deviate from it:

  1. To save processing time. If long calculations have to be repeatedly done with existing fields, the calculation could be done once and added to the database to save time (while technically introducing redundancy).
  2. To save space. If there are millions of records but only a few of them would have a duplicated field, it would cost a great deal of space to add a new table to remove the redundancy, while any potential problems could simply be handled by the software for the few exceptions.
  3. To increase productivity. Excessive numbers of tables with complex relationships can require so many joins that the system becomes difficult to use and maintain, while simplifying the table structures could make the processes simpler to understand and maintain by developers.

  [*There are obviously trade-offs that must be considered, but there should be clear justifications if third normal form is to be deviated from.*]

- Foreign keys should be set to cascade updates, but not deletes.

  [*This safety feature requires that all associated data be removed from all tables before a deletion is allowed.*]

- Separate tables do not normally need to be created containing descriptions of all codes used in other tables, unless they are maintained externally by users or are displayed to the user by at least two programs.

  [*Descriptions of all codes should be documented in* `COMMENT` *fields in the database, and codes that are only used internally do not need to be documented twice. If the code is changed in the database,*

the code will also need to be changed in the program, and there is no advantage to updating a description in the database verses updating it in a program, particularly if the user will never see the description. And even in cases where the description is displayed by two different programs, it is acceptable to handled it with shared code. Hard coding descriptions in the program is much quicker than making calls to access the database.]

Naming Conventions

- Table and field names should be descriptive of their purpose and should avoid abbreviations unless they are well-known.

  [*This improves readability, especially those who are new to the database.*]

- Table name should be in snake_case format with lowercase words separated by underscores.

  [*This is the de facto naming convention and distinguishes them from field names.*]

- Field names should be in PascalCase format with the first letter of every word uppercase and the rest of the letters lowercase. Acronyms should also follow this rule instead of being all uppercase.

  [*This allows multiple words to be distinguishable without having to add extra characters between them.*]

- It is preferred that primary keys that are a single field normally be the name of the table followed by "Id", with exceptions made for other well-known fields (i.e. SSN, VIN, etc.).

  [*This provides a way to immediately identify primary keys in a consistent manner across tables.*]

- Avoid using reserved words for table and field names, but if necessary they can still be used by enclosing them in back-tick characters (i.e. `` `index` ``) in queries.

  [*This avoids confusion and improves the readability of the code.*]

Data Fields

- Use `VARCHAR` instead of `TEXT` unless the field is over 65,535 characters.

  [*A `VARCHAR` up to 255 characters uses one byte for the index, and up to 65,535 characters uses two bytes for the index, so be aware of those boundaries when selecting its size. Also, a default value can be assigned to a `VARCHAR` but it cannot be assigned to `TEXT`.*]

- Be careful when using the `YEAR` data type.

  [*It is limited to the range of 1901 to 2155, so it is not appropriate for ancient values, and will increasingly become problematic for future dates.*]

- Be careful when using the `DATE` data type.

[*It only "officially" supports dates from* `1000-01-01` *to* `9999-12-31` *(and also* `0000-00-00`*), but other dates may still work anyway. The JavaScript date functions do not interpret these dates properly, so use the SQL function* `YEAR()`, `MONTHNAME()`, *and* `DAY()` *to pass the correct values as separate fields. Also, if any part of the date is* `00` *the JavaScript date functions return* `NULL`*. Alternatively, set the* `dateStrings: true` *SQL connection parameter to force dates to be returned as strings in the* `YYYY-MM-DD` *format.*]

- `CHECK` constraints should be used on every field whenever possible to improve data quality.

  [*While they do take additional time when saving a record, the improved data quality is well worth it.*]

- Do not use the `ENUM` data type, use `CHECK` constraints instead to require a set of values.

  [*The* `ENUM` *data type is proprietary to MySQL and is not very portable to other SQL versions. The sorting feature in particular does not port either. While an* `ENUM` *field can save space by only using 1 byte of code, the same thing can be accomplished by using a 1 byte CHAR field with a* `CHECK` *constraint.* `CHECK` *constraints do not take up any space, and are more flexible to work with, and are part of the SQL standard so they are more portable.*]

- For boolean values, use `TINYINT` or `CHAR(1)`, but do not use `BIT` or `BOOLEAN`.

  [*All of these only take up one byte of space, but* `BIT` *and* `BOOLEAN` *are not part of the SQL standard and are not portable, but are merely aliases for* `TINYINT` *in MySQL.* `TINYINT` *is also not portable, but has one byte equivalent fields in other SQL versions. Both* `TINYINT` *and* `CHAR(1)` *can be limited to two values using a* `CHECK` *constraint, but are flexible and could easily contain other states beyond* `true` *and* `false`.]

- One-byte codes using `CHAR(1)` or `TINYINT` should generally be preferred over a set of strings that are repeated in a field. A string of one or two words in a small table, however, is acceptable.

  [*This simply saves space, but they must be documented and descriptions provided when appropriate as previously discussed.*]

- Default values for fields should be set to its most common value whenever applicable. Use `NULL` to indicate that a field is never applicable, or that it is applicable but it has not been entered yet (such as when its value is unknown). Use zero for numbers or the empty string `''` for text when those are the known values for the field.

  [*The use of* `NULL` *has not been consistently applied across the industry, but this is the intended interpretation. A possible alternative would be to eliminate* `NULL` *altogether for it does not save any space compared to using zero or the empty string.*]

Queries

- All MySQL reserved words should be upper case.

  [*This distinguishes them from the table names, field names, and other search parameters.*]

- Do not use `SELECT *` in queries unless almost all of the fields are needed.

  [*This programming discipline is analogous to only declaring the variables that you need. Bring in unnecessary fields only makes the query take longer and wastes space.*]

- For `JOIN` clauses, use the `ON` keyword instead of `WHERE` to define table relationships.

  [*This conforms to the ANSI standard and keeps a consistent use of the `WHERE` clause.*]

- Use `WHERE` instead of `HAVING` where either of them could be used.

  [*This keeps a more consistent use of the `WHERE` clause.*]

- Use the `AS` keyword to create aliases to distinguish fields with the same name coming from different tables.

  [*Otherwise JavaScript will overwrite the first field with the second field of the same name.*]

- Use `!=` instead of `<>` to represent the "not equal" relational operator.

  [*`!=` is now more common in other programming languages.*]

- Do not use column numbers in the `ORDER BY` clause, but use the field names instead.

  [*This is much clearer to understand and much easier to maintain when fields are added and deleted.*]

- One space should occur after any comma, colon, or semicolon (if there are other items following it). But there should not be a space before any of them.

  [*This follows the conventions of the English language and helps improve the readability of items that occur in lists.*]

MySQL Reserved Words

ACCESSIBLE
ADD
ALL
ALTER
ANALYZE
AND
AS
ASC
ASENSITIVE
BEFORE
BETWEEN
BIGINT
BINARY
BLOB
BOTH
BY
CALL
CASCADE
CASE
CHANGE
CHAR
CHARACTER
CHECK
COLLATE
COLUMN
CONDITION
CONSTRAINT
CONTINUE
CONVERT
CREATE
CROSS
CUBE
CUME_DIST
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATABASE
DATABASES
DAY_HOUR
DAY_MICROSECOND
DAY_MINUTE
DAY_SECOND
DEC
DECIMAL
DECLARE
DEFAULT
DELAYED
DELETE
DENSE_RANK
DESC
DESCRIBE
DETERMINISTIC
DISTINCT
DISTINCTROW
DIV
DOUBLE
DROP
DUAL
EACH
ELSE
ELSEIF
EMPTY
ENCLOSED
ESCAPED
EXCEPT
EXISTS
EXIT
EXPLAIN
FALSE
FETCH
FIRST_VALUE
FLOAT
FLOAT4
FLOAT8
FOR
FORCE
FOREIGN
FROM
FULLTEXT
FUNCTION
GENERATED
GET
GRANT
GROUP
GROUPING
GROUPS
HAVING
HIGH_PRIORITY
HOUR_MICROSECOND
HOUR_MINUTE
HOUR_SECOND
IF
IGNORE
IN
INDEX
INFILE
INNER
INOUT
INSENSITIVE
INSERT
INT
INT1
INT2
INT3
INT4
INT8
INTEGER
INTERSECT
INTERVAL
INTO
IO_AFTER_GTIDS
IO_BEFORE_GTIDS
IS
ITERATE
JOIN
JSON_TABLE
KEY
KEYS
KILL
LAG
LAST_VALUE
LATERAL
LEAD
LEADING
LEAVE
LEFT
LIKE
LIMIT
LINEAR
LINES
LOAD
LOCALTIME
LOCALTIMESTAMP
LOCK
LONG
LONGBLOB
LONGTEXT
LOOP
LOW_PRIORITY
MASTER_BIND
MASTER_SSL_VERIFY_SERVER_CERT
MATCH
MAXVALUE
MEDIUMBLOB
MEDIUMINT
MEDIUMTEXT
MIDDLEINT
MINUTE_MICROSECOND
MINUTE_SECOND
MOD
MODIFIES
NATURAL
NOT
NO_WRITE_TO_BINLOG
NTH_VALUE
NTILE
NULL
NUMERIC
OF
ON
OPTIMIZE
OPTIMIZER_COSTS
OPTION
OPTIONALLY
OR
ORDER
OUT
OUTER
OUTFILE
OVER
PARTITION
PERCENT_RANK
PRECISION
PRIMARY
PROCEDURE
PURGE
RANGE
RANK
READ
READS
READ_WRITE
REAL
RECURSIVE
REFERENCES
REGEXP
RELEASE
RENAME
REPEAT
REPLACE
REQUIRE
RESIGNAL
RESTRICT
RETURN
REVOKE
RIGHT
RLIKE
ROW
ROWS
ROW_NUMBER
SCHEMA
SCHEMAS
SECOND_MICROSECOND
SELECT
SENSITIVE
SEPARATOR
SET
SHOW
SIGNAL
SMALLINT
SPATIAL
SPECIFIC
SQL
SQLEXCEPTION
SQLSTATE
SQLWARNING
SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT
SSL
STARTING
STORED
STRAIGHT_JOIN
SYSTEM
TABLE
TERMINATED
THEN
TINYBLOB
TINYINT
TINYTEXT
TO
TRAILING
TRIGGER
TRUE
UNDO
UNION
UNIQUE
UNLOCK
UNSIGNED
UPDATE
USAGE
USE
USING
UTC_DATE
UTC_TIME
UTC_TIMESTAMP
VALUES
VARBINARY
VARCHAR
VARCHARACTER
VARYING
VIRTUAL
WHEN
WHERE
WHILE
WINDOW
WITH
WRITE
XOR
YEAR_MONTH
ZEROFILL

HTML/CSS

General

- All web page code should be written in compliance with the HyperText Markup Language (HTML) version 5 and Cascading Style Sheets (CSS) version 3 standards.

  [*Code should be verified through tools such as https://validator.w3.org. These requirements should be updated when newer standards are released.*]

- Web page code should be developed without the aid of any additional web design software.

  [*Web design software usually introduces bloated code and unnecessary statements compared to coding by hand. It also avoids dependency on a third-party program which may not have longevity, portability, or maintainability. This requirement could be repealed if an excellent open source application becomes readily available.*]

- The structure (HTML markup), presentation (CSS styling), and behavior (JavaScript scripting) concerns must be strictly kept apart, and the interaction between them kept to a minimum.

  [*This provides better readability, maintainability, reusability, and scalability. Avoid using any inline scripting or the use of* `onclick` *or* `onload`*, but use event listeners instead.*]

- The HTML, CSS, and JavaScript may normally be kept in the same file, but the CSS and JavaScript should be moved to separate files when the same code can be shared, or when the file size becomes unwieldy.

  [*The separation of concerns still works well within a single file and it is easier to manage the interaction between them all in one place. Always breaking each concern into a separate file just generates an unnecessary number of small files, always requiring the programmer to switch back and forth to see the context.*]

- Web components using custom HTML elements should be used when appropriate.

  [*This provides greater reusability, scalability, and modularity through encapsulation with the Shadow DOM. This is native to the browser and requires no other external framework.*]

- The file extension for all HTML files should be `.html`, not `.htm`.

  [`.html` *has been the tradition extension used across all platforms, while* `.htm` *was only created by Microsoft because their file system could not originally handle longer extensions.*]

Formatting

- One space should occur after any comma, colon, or semicolon (if there are other items following it). But there should not be a space before any of them.

[*This follows the conventions of the English language and helps improve the readability of items that occur in lists.*]

- Single quotation marks should always be used for properties, even if the property is only one word.

  [*While the specification allows for quotation marks to be omitted if the property is one word, it improves readability and consistency to always include them. Double quotation marks are used for strings in most traditional programming languages.*]

- Two spaces should be used for indentation when needed, but not every construct necessarily needs to be indented.

  [*Many a religious war has been fought over the amount of indentation, but two spaces is the minimum needed for decent readability, so the minimum is adopted to save space.*]

- All HTML element names and attributes and CSS selectors and property should be lowercase.

  [*This ensures consistency and compatibility across different browsers and tools, particularly those that are case sensitive.*]

- Text that is displayed should be formatted with one sentence per line.

  [*Since a third-party editor is not being used, this makes it easier to add and subtract content to the web page. This also is helpful for text diffs and version control.*]

Elements

- Every HTML file must at a minimum contain the following tags:

```
<!DOCTYPE html>
<html lang='en'>
<head>
<meta charset='UTF-8'>
<meta name='viewport' content='width=device-width, initial-scale=1'>
<title>Center for New Testament Restoration</title>
</head>
```

  [*These tags ensure the HTML document is standards-compliant, accessible, and responsive.*]

- All custom HTML elements used by web components should begin with a `x-` prefix.

  [*All custom HTML elements must contain a hyphen anyway and adding the prefix makes it clear that it is a web component.*]

Attributes

- There should be no spaces before or after an equal sign that assigns attributes.

  [*This is a well established convention and to do otherwise merely wastes space.*]

- Absolute references can be used for all files starting from the root website directory, but relative references can only be used for files directly above or below the current directory.

  [*Using a relative reference that goes up and down other directory paths is confusing and can quickly get out of sync when directories are moved.*]

- All images should have `alt=` attributes that contain a description of the image.

  [*This makes the website more accessible to the visually impaired and improves the code readability since it is not always clear what the image is from the filename alone.*]

- HTML attributes should be arranged in the following order:

Order	Category	Attribute
1	External files	src, href, data, target, rel
2	Input	type, disabled, checked, readonly, required, autofocus
3	Identification	id, name, value, placeholder
4	Formatting	class, style, height, width
5	Accessibility	title, alt, role, aria

  [*This helps provide consistency across different types of elements.*]

Properties

- CSS properties should be located in the `<style>` tag or kept in a separate file when the properties can be shared.

  [*The use of the* `style=` *attribute violates the separation of concerns and should be avoided.*]

- Native CSS nesting should be used to group rule sets together whenever possible.

  [*This newer standard is now built into all modern browsers and provides additional clarity, encapsulation, and readability.*]

- CSS properties should be arranged in the following order:

Order	Category	Property
1	Positioning	position, top, right, bottom, left, z-index, float, clear, display, visibility
2	Dimensions	width, min-width, max-width, height, min-height, max-height
3	Spacing	margin, margin-top, margin-right, margin-bottom, margin-left, padding, padding-top, etc.
4	Box layout	box-sizing, overflow, overflow-x, overflow-y
5	Border	border, border-width, border-style, border-color, border-radius, outline
6	Background	background, background-color, background-image, background-position, background-size, box-shadow, opacity, filter
7	Typography	font, font-family, font-size, font-weight, font-style, line-height, letter-spacing, word-spacing, color, text-align, vertical-align, text-decoration, text-transform, white-space, direction
8	Interaction	cursor, pointer-events, user-select, appearance, transition, animation, transform

[*This is based on the Concentric CSS convention developed by Brandon Rhodes: https://rhodesmill.org/brandon/2011/concentric-css*]

- `rem` units should be used to define most sizes, but `px` should be used for sizes related to images.

  [`rem` *units scale with the user's default browser font size, whereas* `px` *locks the size to a fixed value.*]

HTML Elements/Attributes

<!DOCTYPE>
<a>
<abbr>
<address>
<area>
<article>
<aside>
<audio>
<b>
<base>
<bdi>
<bdo>
<blockquote>
<body>

<button>
<canvas>
<caption>
<cite>
<code>
<col>
<colgroup>
<data>
<datalist>
<dd>
<del>
<details>
<dfn>
<dialog>

<dl>
<dt>

<embed>
<fieldset>
<figcaption>
<figure>
<footer>
<form>
<h1> to <h6>
<head>
<header>
<hgroup>
<hr>
<html>
<i>
<iframe>
<img>
<input>
<ins>
<kbd>
<label>
<legend>

<link>
<main>
<map>
<mark>
<menu>
<meta>
<meter>
<nav>
<noscript>
<object>

<optgroup>
<option>
<output>
<p>
<param>
<picture>
<pre>
<progress>
<q>
<rp>
<rt>
<ruby>
<s>
<samp>
<script>
<search>
<section>
<select>
<small>
<source>

<style>
<sub>
<summary>
<sup>
<svg>
<table>
<tbody>
<td>
<template>
<textarea>
<tfoot>
<th>
<thead>
<time>
<title>
<tr>
<track>
<u>

<var>
<video>
<wbr>
accept
accept-charset
accesskey
action
align
alt
async
autocomplete
autofocus
autoplay
bgcolor
border
charset
checked
cite
class
color
cols
colspan
content
contenteditable
controls
coords
data
data-*
datetime
default
defer
dir
dirname
disabled
download
draggable
enctype
enterkeyhint
for
form
formaction
headers
height
hidden
high
href
hreflang
http-equiv
id
inert
inputmode
ismap
kind
label
lang
list
loop
low
max
maxlength
media
method
min
multiple
muted
name
novalidate
onabort
onafterprint
onbeforeprint
onbeforeunload
onblur
oncanplay
oncanplaythrough
onchange
onclick
oncontextmenu
oncopy
oncuechange
oncut
ondblclick
ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart
ondrop
ondurationchange
onemptied
onended
onerror
onfocus
onhashchange
oninput
oninvalid
onkeydown
onkeypress
onkeyup
onload
onloadeddata
onloadedmetadata
onloadstart
onmousedown
onmousemove
onmouseout
onmouseover
onmouseup
onmousewheel
onoffline
ononline
onpagehide
onpageshow
onpaste
onpause
onplay
onplaying
onpopstate
onprogress
onratechange
onreset
onresize
onscroll
onsearch
onseeked
onseeking
onselect
onstalled
onstorage
onsubmit
onsuspend
ontimeupdate
ontoggle
onunload
onvolumechange
onwaiting
onwheel
open
optimum
pattern
placeholder
popover
popovertarget
popovertargetaction
poster
preload
readonly
rel
required
reversed
rows
rowspan
sandbox
scope
selected
shape
size
sizes
span
spellcheck
src
srcdoc
srclang
srcset
start
step
style
tabindex
target
title
translate
type
usemap
value
width
wrap