

J3Script 0.9.1 - Programming Update

Class-based programming, concurrent tasks, verified source and IDE execution

J3Script 0.9.1-alpha | Compiler-sourced release supplement | 13 October 2026

This release extends the real J3Script compiler and downloadable runtimes. It is an **alpha**, not a production-security certification or proof that all seven long-term milestones are complete. This supplement takes precedence over earlier 0.8/0.9 claims where the grammar changed.

1. Exact language headers and project behavior

Header	Primary output	Additional explicit project modes
HTML	.html	None
CSS	.css	None
JavaScript	.js	project module
C	.c (optional native binary)	project module
SQL	.sql	None
Python	.py	project console / module / app
PHP	.php	project app / module / plugin / wordpress-app

Headers remain strict and case-sensitive. Project names are separate from language selectors. HTML/CSS/SQL are specialized target languages, not general-purpose runtime VMs.

2. First-class class declarations (standalone targets)

Use **class ClassName** at the top level of a Python, JavaScript, PHP, or C source file. Add **field name default expression** declarations, and define methods with **to methodName parameters**. Within methods **self** is automatically available. Create an instance with **ClassName()** and call its methods using **instance.method(values)**.

```
Python Counter Demo
class Counter
  field value default 0
  to increase amount
    set self.value to self.value + amount
    give back self.value
set counter to Counter()
```

```
print counter.increase(5)
print counter.increase(7)
print counter.value
```

Output: 5, 12, 12 on separate lines. This exact source was compiled and executed for the Python, JavaScript, PHP and C targets. Fields are mutable only if declared; writing an unknown field produces a runtime error. Class constructors currently take no arguments and use declared default values.

3. How classes are represented

Classes compile to record-backed objects with generated constructor and method functions. This is genuine language syntax implemented by the compiler, but it is **not** yet a full object-oriented type system. Runtime method dispatch is statically resolved from constructed variables, without inheritance, interfaces, access modifiers, custom constructors, or assignment of arbitrary methods as values.

Code that accepts a class instance as an untyped function parameter cannot yet call its methods through dot syntax unless the compiler can infer the specific class. An instance can still be passed as a record. Do not assume every Python/JavaScript class feature exists in J3Script.

4. Starting work and awaiting results (Python and JavaScript)

Both supported targets implement **start FUNCTION arguments into handle** and **await handle into result**. Python runs user-defined functions through a bounded thread pool with a separate interpreter context. JavaScript starts Promise-backed async functions and awaits their results. These are real asynchronous task handles, not simply renamed sequential function calls.

```
JavaScript Async Work
to double number
  give back number * 2
to square number
  give back number * number
start double 5 into first
start square 7 into second
await first into answer1
await second into answer2
print answer1
print answer2
```

Output: 10 and 49. The same source with the header changed to Python is also validated and executed. C and PHP deliberately reject start/await at compile time: no backend implementation for those targets exists yet.

Async limitations: no cancellation syntax, no portable locks/channels for task handles, no guaranteed execution order between tasks, and no equivalent support in Python/PHP web application buttons. Python waits up to 20 seconds on each await and has at most four worker threads; completed tasks not awaited may continue until program exit. Generated JavaScript is asynchronous and must be tested in its target environment.

5. Local package management from the command line

Use the new **lock** subcommand to record an exact semantic version, project-relative path, and SHA-256 of a local J3Script module. There are no automatic registry downloads.

```
python -m j3c lock math@1.2.3 math.j3 --project .
```

The command creates or updates **j3-packages.json**. Imported packages are checked for exact version and byte-for-byte SHA-256 integrity. A package file changed without regenerating the lock is rejected during compilation.

Python

```
use package "math@1.2.3"
print double(8)
```

Modules may now declare classes as well as functions. Transitive package solving, a registry client, namespaced symbols, and package publication are not implemented.

6. Web Studio changes

The bundled browser compiler is regenerated from the same editable source. The IDE retains seven real-language options, bold blue keywords, cyan actions, purple types, green strings, gray comments, plain identifiers, multi-file editing, autocomplete, IndexedDB project recovery, diagnostics, and ZIP export.

A new **Run JS** action executes one compiled JavaScript program in a dedicated Web Worker with output displayed in the IDE and a five-second termination limit. The worker cannot access the editor DOM directly. This is a convenience runner **not a hardened sandbox**; only run code you trust. Python/C/PHP execution is not yet available from this Web Studio button.

7. Demonstrated compatibility and acceptance criteria

Test	Observed result
Standalone class construction / field update / methods	Python, PHP, JavaScript, C compiled and ran; outputs match
Asynchronous task handles	Python and JavaScript compiled and ran; matching sample results
Unsupported asynchronous targets	C and PHP reject syntax explicitly
Package pinning	Exact-version, SHA-256 tamper rejection and traversal checks
Existing release compatibility	0.7.1, 0.8, 0.9 targeted suites passed
IDE syntax and generated JS	Node syntax checks and existing autocomplete tests passed

The older v0.1 golden-file regression suite is historical and compares generated source byte-for-byte against pre-0.6 runtimes; it is not a current conformance test. Live destination-host and full browser-interaction acceptance remain outstanding.

8. Incomplete production milestones

Workstream	Still required
Unified language engine	App syntax integration for classes and async; broader cross-runtime error semantics
Standard library and I/O	Portable filesystem/HTTP APIs with capability/permission design
Packages	Namespacing, transitive dependency resolution, registry, signed distribution
Object model	Dynamic dispatch, inheritance/interfaces, constructors with arguments, access control
Async/concurrency	C/PHP backends, cancellation, timeouts and concurrency safety across runtimes
Security/testing	Security audit, malformed-source fuzzing, actual database/WordPress production acceptance
IDE	Full browser end-to-end testing, debugger, language-server-like symbol analysis and more execution targets

Version 0.9.1 is a verified alpha improvement, not J3Script 1.0. The seven-milestone vision is still partially open, and remaining items must not be advertised as implemented.

9. Quick troubleshooting and next steps

Use **python -m j3c check your-file.j3** to validate exact grammar, and **python -m j3c build your-file.j3 --run** for supported standalone runtimes. Keep four-space indentation. A method error may indicate that the compiler cannot infer the receiver class. If C or PHP reports start/await unsupported, this is an intentional target restriction.

Language update & completion audit

These pages document actual changes from J3Script 0.8.0. The complete 0.8 syntax reference follows; its older version headings refer to the baseline, not a second compiler.

The seven real-language headers (unchanged)

HTML, CSS, JavaScript, C, SQL, Python, PHP. Each selects the named output language. Python/PHP remain standalone by default; [project app](#) and [project plugin](#) are explicit specialized output modes, not languages. No legacy headers are accepted.

Shared application expression calls (new)

The Python and PHP database-application VMs now support the 31 pure standalone standard-library functions, list indexing, and function calls nested inside expressions. Filesystem, HTTPS and CLI operations remain unavailable from the web application VM. Runtime execution is bounded.

```
Python Customers
  project app
  to double n
    give back n * 2
  page home
    title "Contacts"
    when button "Calculate" is clicked
      set n to double(sum_numbers([2, 4, 6]))
      show message "Answer: {n}"
```

Version-pinned local packages (new)

use package "helpers@1.0.0" imports only local project source. The exact package name, semantic version, local .j3 path and SHA-256 checksum must be recorded in j3-packages.json. The compiler rejects mismatches and project-root escapes. No remote package installs are performed.

```
Python
  use package "helpers@1.0.0"
  print triple(7)

{
  "schema": 1,
  "packages": {"helpers": {
    "version": "1.0.0", "path": "helpers.j3",
    "sha256": "REPLACE_WITH_64_CHARACTER_HASH"
  }}
}
```

Web Studio local autosave (new)

The browser IDE now stores a project snapshot in IndexedDB on the device, prompts for recovery on reload, and displays a saving state. Source remains local to that browser. Browser quota, private browsing, eviction and user clearance can remove the saved copy: continue exporting source ZIP backups.

Status of the seven requested milestones

Milestone	Tested result / remaining work
1. Unified engine	31 pure built-ins and expression calls added to Python/PHP app VMs. Cross-target semantics still differ in edge cases.
2. Library and I/O	App pure library parity advanced; system/network/file I/O is still target-specific.
3. Modules/packages	Local pinned package manifest and content verification implemented. No network registry/dependency solver.
4. Classes/objects	Existing record objects; native class/method/inheritance grammar NOT implemented.
5. Async/concurrency	No portable async/await and cancellation model. Target-specific background behavior remains.
6. Testing/security	New parity, manifest/security, and IDE-asset checks; no independent penetration audit or live deployment verification.
7. IDE	IndexedDB recovery and existing highlighting/autocomplete/compiler rebuilt; no complete debugger or sandboxed multi-runtime execution.

What was verified

New 0.9 alpha tests check 31-function application expressions and nested calls, PHP helper execution, package version/hash/path rejection, and presence of autosave assets. Original 0.8 and 0.7.1 acceptance suites pass. Full site browser acceptance remains unverified because the sandbox blocks local HTTP navigation. Historic 0.5 app tests contain obsolete headers and cannot be used unchanged as current grammar conformance tests.

Next required engineering gates

A genuinely complete 1.0 would require class semantics (if adopted), uniform async/await, a cross-target host-capability model, full runtime conformance including Unicode/edge cases, robust packages with transitive dependency locking, debugger and safe console execution, and production security and browser/hosting acceptance. These are not claimed complete in this alpha.

J3SCRIPT / VERSION 0.8.0

J3Script

The Complete Language Guide

Seven headers. Implemented features and exact target limits.

A zero-to-advanced curriculum.

An extensive practical guide to HTML, CSS, JavaScript, C, SQL, Python and PHP projects, with a complete official syntax cheat sheet.

Compiler-verified examples

Full keyword reference

13 October 2026 · Source-grounded edition

Contents

Go directly to a target-language chapter or use the reference tables at the back.

New in 0.8: Extended core, JSON and conformance

J3Script 0.8.0

What's new in J3Script 0.8.0

Getting Oriented

Setup and the Web Studio IDE

The Common J3Script Language

Header 1 — HTML

Header 2 — CSS

Header 3 — JavaScript

Header 4 — C

Header 5 — SQL

Header 6 — Python

Header 7 — PHP

Building Real Web Applications — Python and PHP

Cross-Target Rules, Exports and Troubleshooting

Guided Projects — Learn by Building

End-of-Course Assessment

COMPLETE COLOR-CODED CHEAT SHEET

Glossary and Source Notes

J3Script 0.8.0 — Extended General-Purpose Core

Version 0.8.0 adds 14 portable standard functions and four Python-console-only operations to the actual 0.7.1 compiler and four executable runtimes. It preserves the seven exact file headers. This supplement is part of the full implemented-language reference; the earlier chapters below remain applicable where not superseded here.

What's finished and what is not

All seven language headers work and determine the emitted language. Python/PHP are **standalone by default**; database apps require `project app`, WordPress-compatible PHP apps require `project wordpress-app`, and PHP plugins use `project plugin`. There is **no WordPress language** in J3Script.

Exact header	Implemented primary output	Qualification
<code>HTML</code>	HTML and static assets	No console VM
<code>CSS</code>	CSS rules and responsive selectors	No console VM
<code>JavaScript</code>	Shared 31-function core	Host-specific browser or Node APIs
<code>C</code>	Shared 31-function core, native binary if compiler installed	Native compilation and OS/CPU-specific binary
<code>SQL</code>	MySQL/MariaDB SQL statements	No J3Script execution VM
<code>Python</code>	31 shared functions and 4 console-only APIs	Standalone console; database apps use a different VM
<code>PHP</code>	31 shared functions in standalone/plugin runtime	Web apps use a different VM; WordPress is a PHP deployment mode

Core language — reminder

A .j3 source file begins with a language name and optionally a project name. Shared standalone constructs include `set`, `if`, `otherwise`, `repeat`, `for each`, `to`, `give back`, `do`, `task`, `catch`, lists, records, functions, and expressions. Four-space indentation is syntactically significant.

New portable mathematics and collections

Python

```
set scores to [14, 21, 6, 17]
set total to sum_numbers(scores)
set limited to clamp(total, 0, 50)
make person with name "Ada", score 21
set labels to join(keys(person), ",")
print "Total: {total}"
print "Limited: {limited}"
print "Labels: {labels}"
```

This example is valid 0.8.0 source. Its mathematical and collection expressions also work when the first line is JavaScript, C, or PHP. The generated program is different, but the core results in the acceptance suite match.

New function semantics

Function / arity	Meaning
absolute (1 arguments)	Absolute numeric magnitude
floor (1 arguments)	Round down to greatest integer not above value
ceiling (1 arguments)	Round up to least integer not below value
lowercase (1 arguments)	Convert text to lowercase
uppercase (1 arguments)	Convert text to uppercase
trim (1 arguments)	Remove surrounding whitespace from text
length (1 arguments)	Count list items or text characters
power (2 arguments)	Raise a number to a numeric power
modulo (2 arguments)	Remainder using floor-based division
at (2 arguments)	Read a list/text index or record field; negative indexes supported for lists
contains (2 arguments)	Check membership in a list or substring in text; record key membership
range (2 arguments)	Create half-open integer sequence from start to end, capped at 100,000 items
split (2 arguments)	Break text on a nonempty delimiter, producing a list
join (2 arguments)	Combine list values using a text separator

Core functions reject incompatible value types rather than silently coercing them. `round_number` rounds halves away from zero; `list_slice` excludes the ending index; `keys` returns the record's insertion-ordered keys. For complex Unicode text, validate consistency on your target runtimes: the C backend still has lower-level byte-oriented limitations in some operations.

Python-only JSON, HTTPS and CLI

Function	Only supported by	Meaning
<code>json_parse</code>	Python console	Parse JSON text into lists/records/numbers/booleans (Python console only)
<code>json_stringify</code>	Python console	Encode a J3Script value as JSON (Python console only)
<code>http_get</code>	Python console	Retrieve up to 1 MB of HTTPS text; 5 second timeout (Python console only)
<code>arguments</code>	Python console	Return CLI arguments as a list, excluding script path (Python console only)

Use a JSON file to avoid confusing literal braces with J3Script's string interpolation syntax:

```
Python Data Processor
read text from "records.json" into raw
set record to json_parse(raw)
print record.total
write json_stringify(record) to file "records-copy.json"
print length(arguments())
```

For a remote HTTPS API, you may call `http_get("https://example.com/data.json")` and then `json_parse` its result. HTTP failures raise `HTTPError`; the operation requires HTTPS and caps its response at 1 MB with a five-second timeout. It is a simple utility for trusted console applications—not a browser networking API, production HTTP client, isolation boundary, or authorization mechanism. Generated programs run with the host user's permissions. Do not execute untrusted J3Script.

Shared modules

For standalone targets, use module `"helpers.j3"` imports project-local function definitions. A helper's first line must name the same actual target language and include `project module`. Imports reject traversal outside the project and cyclic dependencies. There is no public package registry, dependency solver, language-wide ABI, or generalized app-module system.

Verified conformance and deployment evidence

The included `tools/test-v080.py` executes generated programs for Python, PHP, JavaScript and native C, comparing expected outputs of the 14 new functions, and tests JSON, CLI arguments, HTTPS URL validation, and target-specific compile-time errors. The existing 0.7.1 language-header tests also remain. Production PHP/WordPress hosting, live network access and browser-only IDE interaction require destination verification.

Remaining general-purpose gaps

J3Script has a legitimate lightweight general-purpose core, but **this release does not claim full implementation of every previously proposed feature**. Database-backed app actions still use a different, bounded logic VM; Python console exposes I/O unavailable elsewhere; classes and inheritance, general networking across all backends, a formal dependency/package ecosystem, portable async/concurrency and process isolation have not been implemented. See the release's `compiler/README-0.8.md` and acceptance tests before promising compatibility.

What's new in J3Script 0.8.0

A language selector selects exactly that language

This release deliberately removes the old implicit application defaults. **Python** means **Python and generates an ordinary .py script**. **PHP** means **PHP and generates an ordinary .php script**. This is true whether you select the language in Web Studio or write the header directly in a .j3 file. The other five headers are **HTML**, **CSS**, **JavaScript**, **C**, and **SQL**; all seven generate their corresponding language artifacts. No platform, plugin, website, or database application is inferred merely from a language header.

Select this language	Standalone output when no project kind is declared
HTML	.html (optional page title)
CSS	.css
JavaScript	.js
C	.c (optional separately compiled executable)
SQL	.sql
Python	.py script
PHP	.php script

Specialized projects require explicit syntax. A Python web app requires **project app**; a PHP web app requires **project app**; a WordPress-compatible PHP app requires **project wordpress-app**; a traditional PHP plugin requires **project plugin**; reusable helper functions can use **project module** in supported language targets. **project console** in Python remains accepted, but standalone console is already the default. A WordPress plugin is a PHP project, not a language.

Example: standalone Python (the default)

```
Python
set amount to 10
print amount * 2
```

j3

Example: standalone PHP (the default)

```
PHP
set amount to 10
print amount * 2
```

j3

Example: explicit Python web application

```
Python Customer Manager
  project app
  database contacts
    name as text, required
  page home
    title "Contacts"
```

j3

Example: explicit PHP web application

```
PHP Customer Manager
  project app
  database contacts
    name as text, required
  page home
    title "Contacts"
```

j3

Language dropdown, syntax colors, and autocomplete

When creating a new file in Web Studio, choose one of the seven names. The editor inserts the appropriate language header and **does not silently create an application**. The colored live editor emphasizes structural keywords in bold blue, actions and events in bold cyan, types in purple, strings in green, comments in muted gray, and ordinary identifiers in light text. Those colors and bold weights are editor presentation only; text-only .j3 source compiles identically. Autocomplete suggests available commands and project identifiers, but the compiler is the authority for validity.

Migrating 0.8.0 source

If a Python file previously relied on **Python** Project Name automatically creating a database-backed app, add an indented **project app** after the first line. Files that already used explicit **project app** should retain it. A PHP or Python file containing **database/page** declarations without an explicit application mode is no longer an implicit app. Old pre-0.6 top-level headers (**page**, **script**, **style**, **data**, **program**, **plugin**, **app**) remain invalid as file headers. Back up existing source before migrating and rebuild against 0.8.0.

Getting Oriented

What “complete” means: This handbook documents the syntax and behavior supported by the supplied **0.8.0 source release**, including restrictions, project types, statements, built-in functions, styles, selectors, application features, exports, and major deployment practices. It is not a reference to *every feature of the destination languages* (such as all JavaScript APIs, all CSS features, or the full PHP standard library), nor does it claim that unimplemented language features exist. Every J3Script statement in the examples uses the actual grammar rather than the native destination-language grammar.

How to use this book

J3Script is a high-level language that **compiles source written in .j3 files** into HTML, CSS, JavaScript, C, SQL, Python or PHP deliverables. It is a compiler/transpiler, not a syntax highlighter that replaces a few words. The same structural concepts recur, but **the allowed grammar changes with the selected language header and project setting**. Read the beginning chapters once, then jump directly to the header you are building with.

Learning route: (1) orientation and tooling; (2) common logic; (3) the seven header chapters; (4) server apps; (5) the troubleshooting and deployment chapters; (6) the complete quick-reference appendix. The second PDF is a compact extraction of the cheat-sheet section for day-to-day use.

Notation: CAPITALS in a syntax signature mean a value you supply; brackets in a *syntax description* mean optional material, while actual list literals like `[1, 2, 3]` contain real square brackets. Lines in code panels are source you can type. `--` begins a J3Script comment outside quoted text. Commands inside code panels are displayed **bold blue/cyan**, types **purple**, quoted strings **green**, and comments **muted gray**; this styling is **not part of the required syntax**.

The seven strict language headers

Exact first-word header	Generated artifact	Typical use	Projects / important qualification
HTML	.html	Static webpages	Name/title optional: HTML or HTML "Welcome"
CSS	.css	Site/application styles	CSS selectors and human-readable style properties
JavaScript	.js	Browser events, Node logic	project module for reusable functions
C	.c plus optional executable	Native command-line work	project module for reusable functions
SQL	.sql	MySQL/MariaDB schema and data	Optional database name after SQL
Python	.py by default	Standalone Python	Explicit project app for web apps; project module for helpers

Exact first-word header	Generated artifact	Typical use	Projects / important qualification
PHP	.php by default	Standalone PHP	Explicit <code>project app</code> , <code>project plugin</code> , <code>project wordpress-app</code> , <code>project module</code>

Case is significant. `JavaScript` works as a header; `javascript` does not. Old *file headers* (`page`, `style`, `script`, `program`, `data`, `plugin`, `app`) are rejected. **This does not ban** those words as *internal statements*, for example `page` dashboard inside a Python application, `use script app` inside an HTML source, or `style` inside an app. WordPress is a **platform**, not a language identifier; PHP is the language header used for WordPress-targeted deliverables.

Example header and project setting:

```

Python Sales Calculator
    set amount to 12
    print amount * 3

```

Here `Python` selects the language. `project console` selects an executable console script. `Sales Calculator` is the source project name, not a Python class declaration. A `.j3` source file is **not** an ordinary `.py` file and cannot contain arbitrary native Python syntax.

Your source-to-output mental model

1. **Author:** Write a UTF-8 `.j3` file with the correct language header, four-space block indentation, and supported statements.
2. **Check:** The parser builds a tree, validates structure, checks names/arguments for supported logic targets, and reports source-line errors.
3. **Build:** The selected generator emits standard target files and, for applications, their accompanying runtime/configuration packages.
4. **Run:** Execute the exported program with its actual destination runtime (browser, Node, native executable, Python, PHP/WordPress, or SQL database).
5. **Revise:** Change the `.j3` source and rebuild. Editing generated output directly is usually lost on the next build.

A successful compile is not a deployment test. Running compiled Python is different from activating a generated PHP plugin in WordPress. A browser HTML preview is not a PHP/SQL/C execution environment.

Setup and the Web Studio IDE

Local command line setup

The compiler is written in Python and requires **Python 3.9+** for its CLI. JavaScript's `--run` needs Node.js; C executables require a C toolchain such as GCC or Clang; PHP applications require PHP 8.1+; SQL outputs target MySQL/MariaDB. Extract the source distribution and run from the `compiler/` folder:

```
python -m j3c --help
python -m j3c check examples/general/python-console.j3
python -m j3c build examples/general/python-console.j3 -o build
python build/python-console.py
```

bash

An optional editable installation (`python -m pip install -e .`) creates the `j3c` terminal command in that Python environment. No compiler installation is necessary to run a **generated Python console script** on another machine that already has an appropriate Python interpreter.

Complete CLI command reference

Command or option	Meaning	Important limit
<code>python -m j3c check FILE [FILE...]</code>	Parse and validate without exporting	Catches syntax/known semantic faults, not every runtime error
<code>python -m j3c build FILE [FILE...]</code>	Compile input files	Accepts multiple paths and wildcard patterns
<code>-o DIR / --out DIR</code>	Put output in a directory	Avoid output collisions
<code>--run</code>	Execute built Python console, JavaScript or C	Browser-dependent JS events do not work in Node
<code>--cc PATH</code>	Choose C compiler	C target only
<code>--backend python php wordpress all</code>	Application export override	Explicit Python/PHP project header controls relevant output; primarily useful for app builds
<code>python -m j3c serve FILE</code>	Locally serve Python/PHP web app	Development, not production hosting
<code>--host ADDRESS</code>	Address for local development server	<code>0.0.0.0</code> exposes it to permitted network peers
<code>--port N</code>	Local server port	Default 8000

IDE use and source safety

The Web Studio includes a multi-file editor, uploads, source ZIP imports/exports, syntax diagnostics, syntax colors, autocomplete, and downloads. `Check all` reports compiler issues; `Build all` creates actual artifacts.

Code coloring **does not change compilation**. The recommended syntax theme is: **bold blue** for keywords; **bold cyan** for actions/events; **purple** for types; **green** for strings; **gray** for comments; **white** for names.

Autocomplete is a suggestion tool, not a guarantee that a statement is legal in the current project. Always compile. Tab can accept a suggestion or indent; Enter can accept a suggestion or preserve/add indentation; Ctrl+Space requests suggestions; Escape closes them. On mobile, examine the whole completed line before building.

The browser Studio runs the Python compiler in the visitor's browser using Pyodide, which may load remotely; it does not keep the compiler source secret from visitors. It does **not** run generated Python/PHP/C/SQL apps on the preview tab. Projects should be saved as ZIPs; do not assume permanent autosave/cloud project history. Keep proprietary source distributions out of `public_html`.

The Common J3Script Language

Source structure, comments, names, and indentation

Most J3Script files start with one exact language header, optionally a `project` setting, then nested blocks. Four spaces = one level of nesting; tabs as leading indentation are rejected. Use one logical statement per line. A comment begins with `--` when outside a string, provided it begins a line or follows a space.

j3

```
Python Everyday Syntax
-- Lines indented by four spaces belong to this project.
set product-name to "J3Script" -- A hyphen can be inside a name.
set subtotal to 10
set discount to 2
print subtotal - discount -- Spaces are necessary for subtraction.
```

Names begin with a letter or underscore; later characters can include letters, digits, underscores and hyphens. A name like `total-cost` can be parsed as **one identifier**, so write `total - cost` with spaces for subtraction. The compiler rejects unknown names in many standalone logic expressions; it does not provide unrestricted runtime reflection.

Values and data types used in logic

Value kind	Example	What it means
Number	7, 19.95, -3	Numeric value; numeric operators require suitable operands
Text	"Hello"	Quoted string; escape literal quotes when needed
Boolean	yes, no	True/false values
No value	nothing	Absence; different from 0 and ""
List	[2, 4, 6]	Ordered elements, zero-based indexed access in 0.7 standalone targets
Record	make lead with name "Ada", score 10	Named fields, read with lead.name or lead["name"]
Function result	square(5)	0.7 expression call in standalone logic targets

Important: `email`, `money`, `date`, `url`, `choice of`, `link to`, etc. are declarations for **specific SQL/application grammars**; they are not universal compile-time type declarations for standalone `set` variables. J3Script 0.8.0 does not expose a complete static type system or arbitrary user-defined classes.

Variables and assignments

j3

```
Python Variables
project console
set name to "Ada"
set quantity to 3
set unit-price to 19.99
set total to quantity * unit-price
add 2 to quantity
subtract 1 from quantity
multiply total by 2
divide total by 2
print "Customer: {name}; total: {total}"
```

`set NAME to EXPR` assigns. `add EXPR to NAME` adds numbers **or appends an item to a list**. `subtract EXPR from NAME`, `multiply NAME by EXPR`, and `divide NAME by EXPR` are separate update statements. Do not substitute conventional Python `+=`, `=`, or JavaScript `let` syntax into `.j3` code.

Expressions and operator precedence

An expression calculates a value from variables, literals, arithmetic, operators, and permitted function calls. The standalone expression parser supports parentheses, collection/index access, and Boolean conditions.

Family	Operator / construct	Example
Arithmetic	<code>+, -, *, /</code>	<code>subtotal + tax, price * quantity</code>
Comparisons	<code>=, !=, <, <=, >, >=</code>	<code>total >= 50</code>
Boolean	<code>and, or, not</code>	<code>is-member and not expired</code>
Grouping	<code>(...)</code>	<code>(a + b) * 2</code>
Field access	<code>.field</code>	<code>customer.name</code>
Index access	<code>[INDEX]</code>	<code>scores[0], scores[-1]</code>
Function call	<code>NAME(ARG, ...)</code>	<code>uppercase(trim(name))</code>
Text interpolation	<code>"Hello {name}"</code>	Inserts the value of <code>name</code> into quoted text

Arithmetic multiplication and division bind more tightly than addition/subtraction. `not` applies before comparisons in the compiler's parser structure, and comparison expressions combine using `and`, then `or`. **Use parentheses** when multiple operator families interact; this is clearer and more portable. `=` is a *comparison*, not assignment. `set` performs assignment. Division by zero raises an error rather than producing a safe result.

Conditions and branching

j3

```
Python Customer Decision
project console
set total to 75
if total >= 100
  print "Large order"
otherwise if total >= 50
  print "Standard order"
otherwise
  print "Small order"
```

The condition following **if** must evaluate to a truth value under the relevant runtime's rules. **otherwise if** checks the next condition only if earlier branches did not match. **otherwise** is the default path and has no condition. Indent the branch bodies. Define a variable before a branch if you expect to read it afterward.

Loops: counted, conditional, and each item

j3

```
Python Three Loop Styles
project console
repeat 3 times
  print "Hello"
set count to 0
repeat while count < 3
  add 1 to count
  print count
for each item in ["north", "south", "west"]
  if item = "south"
    skip
  print item
```

repeat N times evaluates a count; **repeat while** CONDITION repeats while the test holds; **for each** NAME **in** LIST traverses values. **stop** exits the nearest loop and **skip** advances it to the next iteration. Always change a variable in a **repeat while** loop so it can stop. Infinite loops or huge lists can consume execution budgets. **for each** in a JavaScript task marked parallel behaves differently from C/Python sequential logic.

Lists and records

j3

```
Python Collections
project console
make list leads
add "Ada" to leads
add "Linus" to leads
print leads.size
print leads.first
print leads.last
print leads[0]
make agent with name "Grace", score 11
print agent.name
print agent["score"]
```

`make list` NAME starts an empty list; `[ ... ]` is a literal list. `.size`, `.first`, `.last` are supported list attributes; indices are **zero-based** and negative indices count backward in the 0.7 standalone Python/JS/C logic targets. `record["field"]` works in those standalone targets. A record is not a class instance; arbitrary field mutation and full object inheritance are not part of 0.7.

Defining functions, arguments, returns and recursion

j3

```
Python Reusable Functions
project console
to square x
    give back x * x
to double x
    give back x * 2
set value to square(double(4))
do square 7 into other
print "{value}, {other}"
```

A standalone function is declared at the top level with `to` NAME [parameters] . Separate multiple parameter names with commas. `give back` EXPR ends the function with a result. A function's parameters and local values are its own scope; do not rely on reading caller-local variables. `do` NAME ARG, ARG `into` RESULT is the traditional statement call; NAME(ARG, ...) is the 0.7 expression call for supported standalone targets. The compiler checks missing function names and argument counts when it can. The Python console runtime limits recursion to **64 levels**.

A recursive function must have a stopping condition. For example, `if n <= 1` followed by `give back 1` is a base case. Avoid recursion without termination and use loops when possible for long repeated work.

Modules: reusable source files

A module is a separate `.j3` file with the **same language header** as its importer, `project module`, and only top-level function definitions or further module imports.

File `helpers.j3`:

```
Python Helpers
  project module
  to triple n
    give back n * 3
```

j3

File `main.j3`:

```
Python Module Consumer
  project console
  use module "helpers.j3"
  print triple(7)
```

j3

Keep imported modules inside the project directory; the compiler rejects directory traversal, import cycles, and more than 32 distinct imported modules. Imports are folded into the generated output; the destination doesn't have to resolve `.j3` modules at runtime. Module projects are supported for **Python, JavaScript, C and PHP traditional-plugin logic**, not for HTML/CSS/SQL. Module sources do not run as standalone programs.

Error handling, tasks, and rules

```
Python Safe Calculation
  project console
  task calculation
    set amount to 12
    divide amount by 0
    catch DivisionByZero as problem
      print "Problem: {problem.message}"
```

j3

task NAME creates a protected block. **fail** ErrorName "message" raises an error; **catch** KIND **as** problem handles a matching kind; **catch any as** problem catches supported recoverable exceptions. An error record exposes kind and **message**. Error names can differ between standalone and server-application VMs; check the exact backend. Catches do not magically roll back file writes or external actions.

Rules and selectors used for execution policy

j3

```
Python Rules Demo
project console
rules
  task: on error log
  task.safe: on error ignore
task example, safe
  print "Attempting work"
```

Execution selectors: `task` matches every task; `.safe` matches a label; `#name` matches a named block; `task.safe` matches the task kind plus label; `task #name` selects a matching descendant by ancestry. CSS-like specificity determines the effective policy: inline overrides block name `#...`, which outranks class `.label`, which outranks kind, with later rules winning equal ties.

Policy	Supported forms	Important differences
<code>on error</code>	<code>pass up</code> , <code>log</code> , <code>ignore</code> , <code>stop</code> , <code>retry N</code>	In app logic <code>retry</code> is bounded 0-5; not every target enforces every policy identically
<code>run</code>	<code>in order</code> , <code>in background</code>	Background execution is not permitted by Python console/app grammar; C effectively runs it sequentially
<code>timeout</code>	<code>N second(s)</code>	Application VM has enforced budget; some older standalone targets accept rules without enforcing limits
<code>memory</code>	<code>free at end</code> , <code>keep</code>	Not a general memory sandbox; console rejects unsupported persistent-memory rules
<code>log</code>	<code>on</code> , <code>off</code>	Enables execution logging where runtime supports it

Shared rules may be imported using `use rules "shared.rules"`. Inline policy spelling can appear after commas on a task header, such as `task import, on error log`. Different runtimes have materially different security/resource behavior; see the limitations chapter.

The 31-function standalone standard library (0.8.0)

These functions are implemented in standalone **JavaScript**, **C**, and **Python console** logic and in the **traditional PHP plugin logic runtime**. They are **not automatically available** inside Python/PHP/WordPress database-application action logic.

Function	Inputs	Returns / effect	Example
<code>absolute(x)</code>	Number	Absolute value	<code>absolute(-12)</code>
<code>floor(x)</code>	Number	Rounded down	<code>floor(2.9)</code>
<code>ceiling(x)</code>	Number	Rounded up	<code>ceiling(2.1)</code>
<code>power(x, y)</code>	Numbers	x to exponent y	<code>power(2, 5)</code>
<code>modulo(x, y)</code>	Numbers	Numeric remainder	<code>modulo(11, 3)</code>

Function	Inputs	Returns / effect	Example
<code>lowercase(text)</code>	Text	Lowercase text	<code>lowercase("HELLO")</code>
<code>uppercase(text)</code>	Text	Uppercase text	<code>uppercase("hello")</code>
<code>trim(text)</code>	Text	Remove edge whitespace	<code>trim(" Hi ")</code>
<code>length(value)</code>	Text or list	Number of characters/items	<code>length([1, 2])</code>
<code>at(value, index)</code>	Text/list/record, index/key	Selected element/field	<code>at([5, 8], 0)</code>
<code>contains(collection, value)</code>	Text/list/record, value	Membership test	<code>contains([1, 2], 2)</code>
<code>range(start, end)</code>	Integers	List of integers; end exclusive	<code>range(1, 4)</code>
<code>split(text, delimiter)</code>	Two strings	List of parts	<code>split("a,b", ",")</code>
<code>join(list, delimiter)</code>	List, text separator	One text string	<code>join(["a", "b"], "-")</code>
<code>replace(text, find, with)</code>	Three strings	Text with matches replaced	<code>replace("ab", "a", "A")</code>
<code>to_text(value)</code>	Supported value	Text representation	<code>to_text(27)</code>
<code>to_number(text)</code>	Numeric text/value	Number	<code>to_number("42")</code>

These are **built-in functions**, not language keywords. `range(start, end)` excludes `end`, supports descending ranges, and caps created lists at 100,000 elements. An out-of-range `at(...)` access raises an error. Invalid types are errors; do not expect JavaScript-style implicit coercion across backends. UTF-8 handling and edge-case parity across generated runtimes are not yet a fully standardized language contract.

j3

```

Python Library Practice
project console
set names to split("Ada,Grace,Linus", ",")
print join(names, " | ")
print uppercase(trim(" hello "))
print power(2, 4)
print replace("J3 is good", "good", "great")
print at(range(3, 6), 1)

```

Header 1 — HTML

What the HTML header means

Write `HTML "Page Title"` at the start of a source file. J3Script's HTML generator emits an actual HTML5 document with a document title, UTF-8 encoding, responsive viewport, and escaped HTML content. Nested human-readable **element kinds** become HTML tags. This grammar is for **static pages**; it does not query a database, create server storage, or send email merely because you include a form.

```
HTML "Justice Homes"
  description "An example real estate homepage"
  use style home-style
  use script home-events
  header
    title "Find Your Next Home"
    nav
      link "Listings" to "listings.html"
      link "Contact" to "#contact"
  main
    section listings
      card, featured
        heading "Mountain Home"
        image "mountain.jpg" "A home overlooking the mountains"
        text "Three bedrooms and scenic views."
        link "Details" to "details.html"
    section contact
      subtitle "Ask a Question"
      form inquiry
        input email required "Email address"
        button submit-inquiry submit "Send"
  footer
    text "Example J3Script website"
```

j3

`use style home-style` emits a stylesheet link to `home-style.css`; `use script home-events` emits a deferred JavaScript tag pointing to `home-events.js`. **These statements do not build those files automatically.** Create corresponding `.j3` files with **CSS** and **JavaScript** headers, compile them too, and place the results next to the HTML file.

All built-in element kinds

J3Script element	Generated HTML element	Notes
<code>header</code> , <code>footer</code> , <code>nav</code>	<code><header></code> , <code><footer></code> , <code><nav></code>	Site-level semantic regions
<code>section</code> , <code>main</code> , <code>article</code> , <code>aside</code>	Same-named HTML tags	Layout/semantic sections

J3Script element	Generated HTML element	Notes
<code>title</code> , <code>subtitle</code> , <code>heading</code> , <code>small-heading</code>	<code><h1></code> , <code><h2></code> , <code><h3></code> , <code><h4></code>	Headings; one main h1 is a sound convention
<code>text</code> , <code>quote</code> , <code>bold</code> , <code>note</code>	<code><p></code> , <code><blockquote></code> , <code></code> , <code><small></code>	Text content
<code>image</code> , <code>video</code>	<code></code> , <code><video></code>	File path in first quoted argument
<code>link</code> , <code>button</code>	<code><a></code> , <code><button></code>	Link destination / button text
<code>list</code> , <code>numbered</code> , <code>item</code>	<code></code> , <code></code> , <code></code>	Put items beneath their list
<code>form</code> , <code>input</code> , <code>area</code> , <code>label</code>	<code><form></code> , <code><input></code> , <code><textarea></code> , <code><label></code>	Forms need functional JavaScript/server integrations
<code>line</code> , <code>table</code> , <code>row-of</code> , <code>cell</code>	<code><hr></code> , <code><table></code> , <code><tr></code> , <code><td></code>	Basic dividers and tabular structure
Other names, e.g. <code>card</code> or <code>stack</code>	<code><div class="card"></code>	Custom container kinds use a class

The first **unquoted** name after an element kind generally becomes its HTML `id`: `section contact` creates `id="contact"`. Labels following commas become classes: `card, featured` creates a `card` class and a `featured` class. The CSS selector `card.featured` can then target that container.

Element-level syntax catalogue

Syntax	Meaning
<code>title "TEXT"</code>	h1 containing text
<code>heading "TEXT"</code>	h3 containing text
<code>section ID</code>	<code><section id="ID"></code> container
<code>card, CLASS</code>	custom <code><div class="card CLASS"></code>
<code>image "SOURCE" "ALT"</code>	image with alternative text
<code>video "SOURCE"</code>	playable video with controls
<code>link "TEXT" to "URL"</code>	anchor with destination
<code>input email required "PLACEHOLDER"</code>	email-type input, required
<code>input contact-email email</code>	explicit ID/name and input type
<code>area MESSAGE-ID "PLACEHOLDER"</code>	textarea
<code>button SEND-ID submit "TEXT"</code>	submit-type button
<code>line</code>	horizontal rule
<code>use style NAME</code>	link output stylesheet
<code>use script NAME</code>	link output browser script
<code>description "TEXT"</code>	meta description

Recognized special input types: `email`, `password`, `number`, `tel`, `date`, `url`, `search`, `color`, `file`. Non-special inputs default to text. A static `file` input does not create upload processing or file storage. Arbitrary low-level HTML attributes are **not** all exposed through J3Script's page grammar; inspect generated HTML for unsupported integration requirements.

How to connect HTML to CSS and JavaScript

Save the preceding HTML sample as `home.j3`. Save a CSS file as `home-style.j3` and a JavaScript file as `home-events.j3`. Compile all three to one output folder; the HTML references the generated filenames. To check an event, match an ID explicitly: if JavaScript says `when #contact is clicked`, the HTML must declare a matching element ID. The compiler does not automatically resolve IDs across independently compiled files.

Real-world production checks: Use HTTPS. Confirm image paths, output filenames, proper headings, responsive layout, valid forms and links, and actual mail/backend handling for submissions. Previewing a static page in the IDE does not prove that a form has saved anything.

HTML hands-on exercise

Build a two-section landing page with `header`, `nav`, a `card`, `featured`, an image with alt text, and a `form` containing an email input. Compile the result and locate its `id` and `class` attributes in the generated HTML. Change a section identifier and update the anchor link to match.

Header 2 — CSS

The CSS header and selector language

A file that begins with `CSS` compiles to a **real .css file**. Styles can be expressed on one line after a colon, with comma-separated properties, or as child statements indented under a selector. Selectors borrow the HTML kinds from the previous chapter: `page` maps to `body`, `title` maps to `h1`, `card` maps to `.card`. `#id` and `.class` selectors work as expected. This is a defined subset of CSS selector grammar, **not** a complete native CSS parser.

CSS

```
page: background #f7f9fd, color #20314c, font "Inter"
header: background #10233e, color white, padding 24
title: size 38, bold
#listings: layout grid 2, gap 22
card
  background white
  padding 18
  radius 12
  shadow on
card.featured: border 2 #245bd7
button: background #245bd7, color white, pointer
button when hovered: background #12345c
on phone
  #listings: layout column
  title: size 29
```

j3

Selector inventory and examples

Selector spelling	What gets targeted	Example
<code>page</code>	body element	<code>page: background white</code>
<code>title</code> / <code>subtitle</code> / <code>heading</code>	<code>h1</code> / <code>h2</code> / <code>h3</code>	<code>title: size 34</code>
<code>card</code>	class of custom card elements	<code>card: padding 12</code>
<code>#contact</code>	element with ID <code>contact</code>	<code>#contact: margin 24</code>
<code>.featured</code>	elements with class <code>featured</code>	<code>.featured: bold</code>
<code>card.featured</code>	a card with featured class	<code>card.featured: border 1 blue</code>
<code>nav link</code>	links descended from <code>nav</code>	<code>nav link: color white</code>
<code>button when hovered</code>	hover pseudo-state	<code>button when hovered: background black</code>
<code>input when focused</code>	focus pseudo-state	<code>input when focused: border 2 blue</code>

Selector spelling	What gets targeted	Example
<code>item when first</code>	first-child pseudo-state	<code>item when first: bold</code>

Supported human-readable pseudo-states: `when hovered`, `when focused`, `when pressed`, `when first`, and `when last`. Standard supported pseudo-class forms already present in selectors may pass through, but do not presume all advanced CSS selector constructs work. Combinator semantics are generally descendant whitespace rather than a full arbitrary selector engine.

Property shortcuts and value normalization

J3Script property	Generated CSS effect	Example
<code>size N</code>	<code>font-size</code>	<code>size 18 → 18px</code>
<code>radius N</code>	<code>border-radius</code>	<code>radius 12</code>
<code>font "NAME"</code>	<code>font-family stack</code>	<code>font "Inter"</code>
<code>border N COLOR</code>	solid border	<code>border 2 red</code>
<code>border none</code>	no border	<code>border none</code>
<code>align center</code>	<code>text-align: center</code>	<code>align center</code>
<code>layout row</code>	flex row with wrapping	<code>layout row</code>
<code>layout column</code>	flex column	<code>layout column</code>
<code>layout grid N</code>	N equal CSS grid columns	<code>layout grid 3</code>
<code>layout center</code>	flex centering on both axes	<code>layout center</code>
<code>shadow on</code> / <code>shadow none</code>	preset shadow / none	<code>shadow on</code>
<code>bold</code> , <code>italic</code>	font weight / style	<code>bold</code>
<code>uppercase</code> , <code>underline</code>	text transformation/decoration	<code>uppercase</code>
<code>hidden</code>	<code>display: none</code>	<code>hidden</code>
<code>rounded</code>	preset radius	<code>rounded</code>
<code>pointer</code>	pointer cursor	<code>pointer</code>
<code>center</code>	center-aligned text	<code>center</code>

Recognized length properties automatically add `px` to plain numeric measurements (apart from zero): `padding`, `margin`, `width`, `height`, `gap`, `max width`, `min width`, `max height`, `min height`, `top`, `right`, `bottom`, `left`, `letter spacing`. `line height` remains unitless when provided as a number. Many other property names are passed through with spaces converted to hyphens, but the compiler does **not** verify that arbitrary CSS names or values are valid for browsers.

Responsive breakpoints

Directive	Generated query	Caution
<code>on phone</code>	<code>max-width: 600px</code>	Also satisfies tablet's maximum width
<code>on tablet</code>	<code>max-width: 900px</code>	Matches phone widths too
<code>on desktop</code>	<code>min-width: 901px</code>	Independent desktop region

The two maximum-width queries **overlap**. Order your rules deliberately. Inspect generated CSS if an element unexpectedly inherits tablet styles at phone width.

Styling generated applications

The same style generator is available as an indented `style` block **inside a Python or PHP application**. For generated app UIs, use runtime CSS classes such as `.j3-main`, `.j3-card`, `.j3-table`, and `.j3-button`, as well as semantic selectors like `title`. This is a second, internal use of `style`, not a valid top-level language header by itself.

CSS troubleshooting

- **Nothing changes:** confirm the HTML file links the built `.css` name, clear caches, and inspect that file in the browser.
- **Selector does not match:** check whether the HTML kind becomes a native tag or a `.class` wrapper. For example `card` means `.card`, not a `<card>` tag.
- **Mobile override fails:** check order of `on phone` and `on tablet` blocks; both match at 600px and below.
- **A shortcut changes unexpected properties:** inspect generated CSS, particularly `layout row` and `layout center`, which emit multiple declarations.

CSS hands-on exercise

Style a three-card layout as a grid, add a `featured` border, an accessible button hover/focus state, and make it one column for phone widths. Verify the generated selectors use `.card` rather than imaginary HTML tags.

Header 3 — JavaScript

Standalone logic and browser events

A **JavaScript**-headed `.j3` file produces `.js` source. Shared J3Script logic supports values, assignments, loops, functions, lists, records, tasks, rules, and the 31-function standalone standard library. Additional **browser-only event/DOM** statements can interact with page elements. The emitted JavaScript can run in Node.js **only when it doesn't depend on the browser DOM**.

j3

```
JavaScript Greeting Button
  when #greet is clicked
    read #visitor into person
    set message to "Hello, {person}!"
    put message into #result
    show #result
```

Match `#greet`, `#visitor`, and `#result` to real IDs in the generated or existing HTML. Compile and link the JavaScript file using an HTML source's `use script NAME` statement. The cross-file IDs aren't statically validated for you.

All JavaScript event spellings

J3Script event header	Browser event
when #button is clicked	click
when #field is changed	change
when #form is submitted	submit
when #target is hovered	mouseenter
when #field is typed in	input
when page loads	page-load handling

A `when` block must contain actions. `when page loads` does not have an element selector. The implementation's selector for DOM statements is a `#id` or `.class`-style selector as accepted by the parser; inspect the emitted code and test browser event matching for your actual elements.

Browser DOM statements

Statement	Effect
read #id into VARIABLE	Read the current control value
put EXPR into #id	Put a value/text into an element

Statement	Effect
<code>show #id</code>	Show element
<code>hide #id</code>	Hide element
<code>when #id is submitted</code>	React to form submission
<code>when page loads</code>	Initialize browser-side behavior

j3

```
JavaScript Form Interaction
  when #signup is submitted
    read #email into submitted-email
    if submitted-email = ""
      put "Please enter an email" into #message
    otherwise
      put "Received: {submitted-email}" into #message
    show #message
```

This form example only changes browser display. It does not send an email, insert into a database, perform server-side validation, or prove that Mailchimp received an address. Production submissions need an explicit integration/backend.

General-purpose JavaScript example

j3

```
JavaScript Invoice Math
  to add-tax amount, rate
    give back amount * (1 + rate)
  set prices to [25, 50, 75]
  set total to 0
  for each price in prices
    add price to total
  print add-tax(total, 0.07)
  print join(prices, ", ")
```

This is a useful Node-friendly example: no DOM. Use `python -m j3c build invoice.j3 --run` when Node is installed. The J3Script source syntax is not native JS syntax; `to`, `set`, and `give back` are compiled away.

JavaScript task and concurrency notes

In JavaScript outputs, background tasks can use asynchronous execution. A `for each` labeled `parallel` may launch work concurrently and await it. The same policy is **not portable** to C, Python console, or server-app actions. Browser event handlers operate on an event loop; native `open` and filesystem behavior depend on Node, not the browser. Use only a trusted, supported environment for filesystem-capable scripts.

JavaScript modules

For shared function definitions, create `JavaScript Shared Logic` with `project module`, then import it from another JavaScript `.j3` using `use module "helpers.j3"`. This is compile-time J3Script function reuse. It is **not** JavaScript `import ... from ...` package syntax or a package manager.

JavaScript project challenge

Build a static HTML page with an email input, a submit button, and a hidden success message. Give the elements IDs. Write JavaScript `.j3` event logic to read the email and reveal a local confirmation. Then explain why this does not guarantee server delivery.

Header 4 — C

C output: compilation into a native executable

The `C` header generates `.c` source. If GCC/Clang (or another compatible C compiler) is available, `j3c build` also attempts to produce a native executable for the current system's OS/CPU. A C binary isn't a browser page or a PHP deployment package; compile it for the actual destination platform.

```
j3
C Native Total
  to double value
    give back value * 2
  set prices to [10, 20, 30]
  set total to 0
  for each price in prices
    add price to total
  print double(total)
```

The above uses shared logic. The 0.7 collection and standard-library operations implemented in the C runtime allow `range`, `join`, `length`, and other listed functions, subject to the runtime's value and memory limits.

Optional C library declarations

```
j3
C Math Library
  use c library "libm"
    to pow a as number, b as number gives number
  do pow 2, 10 into result
  print result
```

`use c library "libm"` declares a native linked library; child `to` signatures describe native functions. The supported signature types are `number`, `int32`, `byte`, `text`, and `nothing`. Native calls must match library name, parameter types and calling ABI, and the destination must supply the binary library. This facility is **not** a sandbox for arbitrary untrusted native code.

Native-specific behavior and limits

- A generated binary runs with the privileges of the process launching it.
- A C build depends on GCC/Clang/link libraries at compile time; binaries are OS/architecture-specific.
- Background blocks in C effectively run sequentially; do not depend on JS asynchronous ordering.
- Native resource handles are checked before C calls, but allocations generally remain until program exit unless explicitly freed; automatic per-block reclamation is not complete.

- `free NAME` is a resource/memory operation in supported standalone logic, **not** a universal garbage-collection command.
- The system's compiler flags and external libraries affect portability. Generated C should be reviewed when extending native APIs.

C modules and reproducible builds

`C` `My Helpers` with `project module` can hold reusable functions. Import with `use module "my-helpers.j3"` from another `C` project. The compiler embeds imported functions in the C output; no external JavaScript module runner is required at execution time. Keep original source and imported modules in version control.

Native program exercise

Build a program that creates `range(1, 6)`, doubles each number, stores the results in a list, and prints them with `join`. Verify output on your local operating system, then find the generated `.c` source and explain where execution depends on a C compiler.

Header 5 — SQL

SQL is for generating schema and queries, not executing them

J3Script's `SQL` header produces MySQL/MariaDB-compatible `.sql` statements. It does **not** connect to a database, execute its output, or maintain migrations during compilation. An optional name in `SQL realestate` emits `CREATE DATABASE/USE`; omit the name when the database is already selected by your hosting control panel.

```
SQL realestate
  table agents
    name text, required
    phone text 20
  table homes
    address text, required
    price money, required
    status choice of "active" or "sold", default "active"
    agent link to agents
  add to agents
    name "Ada", phone "555-0100"
  add to homes
    address "12 Oak Street", price 249900, status "active", agent 1
  find address and price from homes where price < 300000, sorted by price descending, limit 10
```

The relational table referenced by `link to agents` must already exist when the resulting foreign key is created. Ordering definitions matters.

Full SQL table-type reference

J3Script SQL declaration	Generated MySQL/MariaDB type
<code>number</code>	<code>INT</code>
<code>big number</code>	<code>BIGINT</code>
<code>decimal</code>	<code>DECIMAL(12,4)</code>
<code>money</code>	<code>DECIMAL(12,2)</code>
<code>text</code>	<code>VARCHAR(255)</code>
<code>text N</code>	<code>VARCHAR(N)</code>
<code>long text</code>	<code>TEXT</code>
<code>yes/no</code>	<code>TINYINT(1)</code>
<code>date</code>	<code>DATE</code>

J3Script SQL declaration	Generated MySQL/MariaDB type
<code>time</code>	DATETIME
<code>email</code>	VARCHAR(255)
<code>json</code>	JSON
<code>link to</code> TABLE	INT foreign key to target table's <code>id</code>
<code>choice</code> of "A" or "B"	ENUM(...)

SQL column modifiers appear after commas: `key` (primary key), `auto` (auto-increment), `required` (NOT NULL), `unique`, `default` VALUE, and `now` (current timestamp on a datetime/time field). The compiler adds an auto-increment `id` unless you mark a primary key. Declaring `id` yourself without `, key` is an error.

Every SQL data statement

J3Script	SQL generated	Caution
<code>table</code> NAME + indented columns	CREATE TABLE IF NOT EXISTS	Later changes are not migrations
<code>add to</code> NAME + indented row pairs	INSERT INTO	Re-running can duplicate rows
<code>find</code> NAME where CONDITION	SELECT ... WHERE	Filter syntax is restricted
<code>find</code> COL and COL <code>from</code> NAME	Selected columns	and is a column-list join in this context
..., <code>sorted by</code> FIELD	ORDER BY ... ASC	Optional <code>descending</code>
..., <code>limit</code> N	LIMIT N	Restricts rows
<code>change</code> NAME where CONDITION + new pairs	UPDATE ... SET ... WHERE	Review broad conditions
<code>remove from</code> NAME where CONDITION	DELETE ... WHERE	Required where, but broad filters still risky

SQL

```

table sales
  customer text, required
  amount money, default 0
add to sales
  customer "Ada", amount 35.50
find customer and amount from sales where amount >= 20, sorted by amount descending, limit 5
change sales where customer = "Ada"
  amount 40
remove from sales where amount = 0

```

SQL expressions permit literals, field names, comparisons, logical `and/or/not`, and SQL-appropriate expressions. **Text interpolation in SQL string literals is rejected** rather than converted into unsafe executable SQL. This SQL generation grammar is not an arbitrary raw-SQL pass-through and isn't the same thing as an application's `database` grammar.

Safety and deployment

Review the exported `.sql` before importing into phpMyAdmin or a MySQL CLI. Use a test database and backup first. A `change/remove` condition can still match unexpectedly many rows. Named SQL projects may fail on shared hosts that forbid database creation; use an unnamed header and select an existing database.

Migration/versioning, production rollback and general join builders are outside this SQL language's current scope.

SQL practice

Create tables for `agents` and `properties`, add two properties, query properties below a maximum price, mark one sold with `change`, and inspect the exact SQL. Do **not** run destructive queries on live business data while learning.

Header 6 — Python

Python language selection and optional project modes

The **Python** header creates an output built for Python, but the optional project type changes the grammar and deliverable:

Project	First lines	Output	Use case
Default standalone	<code>Python</code> or <code>Python</code> Utility	Independent <code>.py</code>	Terminal tools, calculations, local text-file processing
Explicit console	<code>Python</code> Utility (optional <code>project console</code>)	Independent <code>.py</code>	Optional equivalent of the default
Module	<code>Python</code> Helpers + <code>project module</code>	Validated/reused <code>.j3</code> source	Share compiled functions between console programs
Application	<code>Python</code> CRM + <code>project app</code>	Web application directory	Forms, accounts, persistent records, pages

Do not confuse console with app logic. Console statements use the standalone logic parser and its 31 shared standalone built-ins. Python application buttons use a separate, bounded interpreter. As of 0.8.0, those app buttons do not automatically accept every console built-in or syntax feature.

Python console: complete beginner example

```
Python Commission Calculator
project console
ask "Sale price: " into raw-price
set sale-price to to_number(raw-price)
set rate to 0.03
set commission to sale-price * rate
print "Estimated commission: {commission}"
```

j3

A console script prints to a terminal and waits for text input with `ask`. This example is an arithmetic demonstration, not a full real-estate commission policy; it deliberately omits splits, fees, taxes and brokerage agreements. Run with:

```
python -m j3c check commission.j3
python -m j3c build commission.j3 -o build
python build/commission.py
```

bash

The generated `.py` source runs with Python 3.9+ **without importing the J3Script compiler**.

Python-only console I/O statements

Statement	Meaning	Notes
<code>ask</code> PROMPT <code>into</code> NAME	Read a terminal line	Input initially arrives as text
<code>write</code> EXPR <code>to file</code> PATH	Write UTF-8 text to file	Can overwrite existing content
<code>read text from</code> PATH <code>into</code> NAME	Load whole UTF-8 text file	Relative to process working directory
<code>open</code> "FILE.csv" <code>as</code> ROWS	Standalone existing <code>open</code> operation	CSV is parsed to records; non-CSV sources may become text
<code>print</code> EXPR	Print a value	Standard output

I/O is intentionally constrained by target. The first three statements are Python-console-specific and are not interchangeable with JavaScript browser code, C code, PHP app code, or database-backed application buttons. File paths run under the privileges of the **generated program**, so scripts from untrusted sources are dangerous.

j3

```
Python Local Notes
project console
ask "Write a note: " into note
write note to file "notes.txt"
read text from "notes.txt" into saved
print "Stored: {saved}"
```

`write` overwrites; the release does not define a general append-mode file API. Error handling uses standalone task/failure operations, and filesystem writes are **not transactional**. Never assume a later caught error undoes an earlier file write.

Python console functions and collection workflows

j3

```
Python Text Processing
project console
to normalize label
  give back uppercase(trim(label))
set raw to [" first ", " second ", " third "]
make list cleaned
for each item in raw
  add normalize(item) to cleaned
print join(cleaned, " / ")
print "Count: {length(cleaned)}"
```

Console functions can call other functions in expressions, recurse with a defined base case, import modules, and work with 0.7 collection indexing. The console VM has a **1,000,000-step execution budget** and **64-**

level function recursion guard. These are operational safeguards, not operating-system sandboxing. The standard library's `range` also limits list generation.

A Python module with two reusable functions

helpers.j3

```
Python Reusable Utilities
project module
to net amount, deduction
    give back amount - deduction
to display-name name
    give back uppercase(trim(name))
```

j3

main.j3

```
Python Uses Utilities
project console
use module "helpers.j3"
print display-name(" ada ")
print net(100, 12)
```

j3

For imported modules, the **exact same Python header** and `project module` are required; modules can contain function declarations and `use module` statements, but not top-level `print` calls or unrestricted code. A module import is resolved during compilation, not at runtime as a native Python package.

Python application mode at a glance

A `Python My CRM` file compiles to a **standalone Python script** unless you explicitly request an application. Only `Python My CRM` with an indented `project app` uses J3Script's **application grammar**. Under the root, you may declare `database`, `accounts`, `style`, `rules`, imported rules, functions and `page` definitions. App buttons can validate/save forms and navigate. The generated application contains `app.py`, its runtime, model, schema and static assets. It normally uses SQLite persistent storage and a WSGI-compatible deployment arrangement.

```

Python Contact Manager
project app
database people
    name as text, required
    email as email, unique
page dashboard, home
    title "People"
    show people as table, searchable, editable, removable
    link "Add person" to page new-person
page new-person
    title "New Person"
    form person-form
        input name
        input email
        when button "Save" is clicked
            validate person-form
            save person-form to people
            open page dashboard

```

Build and serve locally with `python -m j3c serve contact-manager.j3`. On first use, the app generates setup information and requires an owner password. Persistent data lives separately from the generated source. A fresh exported ZIP does **not** include all live records from another deployed installation.

Hostinger caveat: Hosting the *static Web Studio* is different from hosting a *generated Python server app*. Python server deployment typically requires a suitable VPS/WSGI arrangement; do not upload `app.py` into `public_html` and expect it to behave like PHP. The generated `passenger_wsgi.py` is only useful where Passenger is genuinely available.

Header 7 — PHP

PHP language selection and optional project modes

A PHP header selects PHP as the generated language. The `project` setting indicates **which deliverable** is being produced; it is not a new programming language.

Source header and setting	Output	Notes
<code>PHP</code> or <code>PHP Utility</code>	Standalone <code>.php</code> program	No application or platform implied
<code>PHP Customer Portal</code> + <code>project app</code>	Standalone PHP server application	Database/forms/accounts/pages
<code>PHP Shop Site</code> + <code>project wordpress-app</code>	PHP application packaged for WordPress	WordPress login/roles, app-specific data tables
<code>PHP J3 Greeting</code> + <code>project plugin</code>	Traditional PHP plugin	WordPress shortcode/actions/filters
<code>PHP Shared Functions</code> + <code>project module</code>	Importable source module	Standalone traditional plugin logic functions

WordPress is not a language. `project plugin` and `project wordpress-app` describe PHP deliverables intended for WordPress; they are not file language headers. `WordPress`, `plugin`, and `WordPress Plugin` cannot be used as J3Script's top-level language word.

Standalone PHP application

A PHP application uses the same **application-language grammar** as Python's app output: databases, pages, forms, buttons, limited app logic, and account settings. It exports `index.php`, runtime, configuration example, CSS/JS assets, schema references, `.htaccess`, and a README. PHP 8.1+ and PDO with the chosen database driver are required.

```

PHP Customer Portal
project app
database requests
  name as text, required
  email as email, required
  message as long text
page home, home
  title "Customer Requests"
  show requests as table, searchable, editable
  link "New Request" to page new-request
page new-request
  title "Contact Support"
  form request-form
    input name
    input email
    input message
    when button "Submit" is clicked
      save request-form to requests
      show message "Request saved"
      open page home

```

For deployment on PHP hosting, copy `config.sample.php` to private `config.php`, choose SQLite or MySQL where your host supports the driver, create a private writable data directory outside the public document root, configure database credentials in the file or host settings, and test setup, authentication and persistence. **Do not put secrets or setup codes into downloadable/public files.** App configuration and persistent data must survive redeployment.

Traditional PHP plugin syntax

The `project plugin` form uses the standalone logic parser plus PHP metadata, WordPress shortcodes, actions and filters. This is distinct from `project wordpress-app`. Plugin logic has access to the 31 shared standalone functions where implemented; database-app button actions are not assumed to share those built-ins.

```

PHP J3 Greeting
project plugin
version 1.0.0
author J3Script Developer
description Adds a simple greeting shortcode
to greeting name
  give back "Hello, {name}!"
shortcode greet name
  do greeting name into result
  give back result
when wp_footer
  print "Generated by J3Script"
filter the_title title
  give back title

```

The generator produces a PHP plugin entry point and WordPress registration code. A shortcode `greet` can be used in WordPress content with its declared attributes; `when wp_footer` registers an action hook; `filter the_title title` takes one parameter and returns a filtered value. The plugin generator prefixes/isolates functions by the plugin slug. Hook argument contracts depend on WordPress and the particular hook.

Traditional plugin statement reference

Statement	Meaning
<code>version</code> VERSION	Header metadata for generated plugin
<code>author</code> TEXT	Plugin author metadata
<code>description</code> TEXT	Plugin description metadata
<code>to</code> FUNCTION PARAMS	Reusable logic function
<code>shortcode</code> NAME [ATTRS]	Register shortcode handler
<code>when</code> HOOK	Register WordPress action hook
<code>filter</code> HOOK PARAM	Register WordPress filter (single value parameter)
<code>give back</code> EXPR	Shortcode/filter/function return value
<code>print</code> EXPR	Emitted text at the hook execution point
<code>rules / use rules</code>	Supported execution policy controls
<code>use module</code> "FILE.j3"	Import reusable PHP module functions

The WordPress shortcode format is **platform API usage**, not a J3Script keyword family or a new source-language target. Generated output needs **WordPress staging activation** before use in production. For traditional plugins, logging and user-visible shortcode behavior depend on the generated runtime and host configuration.

PHP WordPress application export

j3

```

PHP Staff Portal
project wordpress-app
accounts
database tickets
  subject as text, required
  status as choice of "open" or "closed", default "open"
page dashboard, home
  title "Support Tickets"
  show tickets as table, searchable, editable, removable

```

The generator packages a PHP app runtime and WordPress adapter. WordPress supplies authentication, login/logout, recovery and nonce behavior; J3Script app-specific tables use a stable namespace/prefix. Administrators with `manage_options` map to owner; users with `edit_others_posts` map to editor; other signed-in users map to viewer, subject to any `j3script_app_role` extension mapping. `accounts` is not a separate duplicate WordPress authentication database.

The app's shortcode opens the complete application route; it is **not** an inline Gutenberg/Elementor page builder component. Uninstall and deactivation are different: deactivation retains data, and renaming an app can change its table identity. A genuine WordPress activation test on staging is necessary; generating valid PHP is not proof of full WordPress runtime compatibility.

PHP deployment and security notes

- Use PHP 8.1+ and the required PDO driver; verify the host's actual extensions.
- For standalone PHP, configure durable data storage outside `public_html` and check read/write permissions.
- For PHP/WordPress apps, protect session state and CSRF; do not infer correctness from a pretty browser preview.
- Password recovery on standalone PHP uses the host's `mail()` configuration, not Python's SMTP configuration by magic.
- Plugins and apps can contain generated runtime code; inspect their PHP before handing source to a third party.
- Live PHP/MySQL deployment and WordPress activation were **not fully verified** by the 0.7 acceptance suite.

Building Real Web Applications — Python and PHP

Application grammar: the shared model

This chapter applies to Python NAME + project app and PHP NAME + project app application projects, including PHP project wordpress-app. It does not apply to Python console source. Most application declarations are nested under the language header:

j3

```
Python Mini CRM
  project app
  accounts
  database leads
    name as text, required
    email as email, unique
  page dashboard, home
    title "My Leads"
  show leads as table, searchable, editable, removable
```

The important app-level declarations are accounts, database, page, style, rules, use rules, and app-level to function definitions. The app defines a declarative data and page model; buttons execute bounded server-side logic for calculations and form actions. It is not a free-form Python/PHP program body.

Database schema types, options and relationships

Type spelling in app database	Meaning / validation
text	Text up to 255 characters
long text	Multi-line text up to 10,000 characters
email	Address-shaped string; not guaranteed deliverable
number	Whole number
decimal	Number with up to eight fractional places
money	Monetary number with up to two fractional places
yes/no	Boolean
date	Date value, e.g. YYYY-MM-DD
time	Date-time value, e.g. ISO-like timestamp
phone	Phone-shaped text with at least seven digits
url	HTTP/HTTPS URL

Type spelling in app database	Meaning / validation
<code>choice</code> of "A" or "B"	One of listed values
<code>link to</code> OTHER-TABLE	Reference to a record in a declared table

App database fields follow `FIELD as TYPE, required, unique, default VALUE` where appropriate. Every stored record gets `id`, `created_at`, and `updated_at` automatically. `required` enforces a value; `unique` enforces a unique value; `default` supplies a fallback. Starting `add ...` rows seed a new table, **not** repeated synchronized changes to existing production records. Field names normalize for database storage; avoid names that would collide after hyphens become underscores.

j3

```
Python Agent Records
project app
  database agents
    full-name as text, required
    email as email, unique
  database homes
    street as text, required
    price as money
    agent as link to agents
    status as choice of "active" or "sold", default "active"
    add street "12 Oak", price 240000, status "active"
page dashboard, home
  title "Properties"
  show homes as table with street and price and status
```

`link to` stores a foreign record ID, presents a selection control and checks that the referenced item exists. Deleting a referenced record can clear a nullable link rather than cascaded-delete dependent records. This is **not** a general joined-query language or per-customer record-security system. App types differ from the SQL-target types: app fields do **not** accept SQL-only `json`, `big number`, `text N`, `key` or `auto` syntax.

Pages, titles, content, navigation and visibility

App page statement/option	What it does
<code>page NAME</code>	Declare application screen/route
<code>page NAME, home</code>	Mark entry page
<code>page NAME, public</code>	Allow anonymous access
<code>page NAME, roles owner and admin</code>	Limit to these roles (owner always allowed)
<code>title "TEXT"</code>	Main title
<code>subtitle "TEXT"</code>	Secondary title
<code>heading "TEXT"</code>	Smaller heading
<code>text "TEXT"</code>	Body paragraph
<code>link "TEXT" to page NAME</code>	Navigation link to a declared page

App page statement/option	What it does
<code>show</code> TABLE <code>as</code> <code>table</code>	Display stored records as table
<code>show</code> TABLE <code>as</code> <code>list</code>	Display stored records as list
<code>show</code> <code>count</code> of TABLE	Show record count

A page is private by default. Without a `roles` restriction, all signed-in account roles can access it; `public` allows guests. `public` and `roles` cannot be combined. A public form may accept anonymous writes, so a public page deserves a deliberate privacy/security review. Hiding a table column **does not** create access control.

Records display, filters, search, paging and CRUD

j3

```
Python Lead Review
project app
database leads
    name as text, required
    status as choice of "new" or "won", default "new"
    score as number, default 0
page dashboard, home
    title "New Leads"
    show leads as table with name and status and score, where status = "new" and score >= 5,
    page size 20, searchable, editable, removable
```

Display option	Meaning / limit
<code>with</code> <code>name</code> and <code>status</code>	Selected visible columns; not a permission boundary
<code>where</code> FIELD OP LITERAL	Source-side field filter
<code>and</code>	Combine literal field filters; no OR grammar
<code>contains</code>	Supported on suitable text-like fields
<code>searchable</code>	Search shown text-like data
<code>editable</code>	Enable built-in edit route/form
<code>removable</code>	Enable deletion control
<code>page</code> size N	Pagination size 1-1000
<code>limit</code> N	Limit results when no page size overrides it
<code>oldest</code> <code>first</code>	Oldest-first display order

Filters support `=`, `!=`, `<`, `<=`, `>`, `>=`, and `contains` for suitable types. Filter literals are bound as SQL parameters; invalid field/operator combinations are diagnosed. The view and search term combine; page navigation maintains the search term. Two displays of the **same database** can share query parameters, so put them on different pages for independent searches. Source filters do **not** enforce row ownership or tenants.

Forms, validation, inputs and buttons

j3

```
Python Contact Form
project app
database contacts
    name as text, required
    email as email, required
    message as long text
page dashboard, home
    title "Contacts"
    show contacts as table, searchable
    link "New contact" to page create
page create
    title "New Contact"
    form contact-form
        input name, label "Full name", placeholder "Ada Lovelace"
        input email
        input message
        when button "Save" is clicked
            validate contact-form
            save contact-form to contacts
            show message "Contact added"
            open page dashboard
```

A form's input can inherit its type and **required** flag from the matching database field when the form saves to that table. Explicit type options must agree with the target field. Inputs can specify **label** "...", **placeholder** "...", **required**, and an explicit type where allowed. If a form never saves to a database, explicit input types may be required.

Button actions execute in order: **validate** FORM, **save** FORM **to** TABLE, **show message** EXPR, **open page** NAME. **save** always validates server-side and inserts a **new record**, not an update of an arbitrary existing record. Built-in **editable** is how the current grammar exposes editing. Navigation should come **last**, because later actions cannot run after **open page**. The server enforces field types and constraints even when client/browser validation is bypassed.

Server-side calculations and app functions

j3

```
Python Score Helper
project app
to double value
  give back value * 2
database applicants
  name as text, required
  score as number, default 0
page home, home
  title "Applicants"
  form score-form
    input name
    input score
    when button "Save score" is clicked
      validate score-form
      do double input.score into result
      if result > 100
        set result to 100
      put result into input score
      save score-form to applicants
      show message "Saved score {result}"
```

`input.FIELD` exposes a submitted form value; validate it before calculations so the value is typed. `put EXPR into input FIELD` modifies the pending value; the subsequent save validates it again. App-level functions can compute and return values, but **form operations, navigation, and input mutation require the actual button context**. `user.id`, `user.email`, and `user.role` expose identity fields in app logic, but no password material. The app VM has more restrictive statements and runtime limits than Python console or JavaScript standalone programs.

Error handling and application transaction behavior

App button database changes are part of a transaction. An uncaught error rolls back the entire button; a protected `task` may roll back to a savepoint before running catch handling or a bounded retry. `fail ErrorName "message"` and `catch KIND as problem` work with **app-runtime error kinds**, which can differ from legacy standalone errors. Catching a recoverable error does not undo outside effects you might add in future integrations.

Application policy/limit	Current 0.7 behavior
<code>on</code> error pass up / <code>stop</code> / ignore / log / retry N	retry N bounded 0-5
timeout N seconds	Applies as app policy within VM constraints
run <code>in</code> order	Supported
run <code>in background</code>	Rejected for transactional app logic
memory <code>free at</code> end	Permitted; persistent app variables rejected
Per-request steps	10,000 statement/expression steps

Application policy/limit	Current 0.7 behavior
Per-request deadline	2 seconds
Nested function calls	32
Individual list/text safeguards	10,000 items/characters at checked values
Numeric magnitude safeguard	10 ¹⁵

These guards are **not** comprehensive system-level resource isolation. Long-running imports, streaming HTTP workloads and arbitrary background jobs belong in additional infrastructure.

User accounts, authentication and roles

Add `accounts` directly under the app header for **multi-user Python/standalone PHP apps**. Without it, they retain single-owner access. The initial owner sets up the application with a private setup code and a password of at least 10 characters. There is no open public self-registration flow in the current generated app UI.

Role	Authority
<code>owner</code>	All pages, all account-management privileges, including admins
<code>admin</code>	Manage editors/viewers; cannot change the owner or create another admin
<code>editor</code>	Write on authorized pages declaring appropriate operations
<code>viewer</code>	Read accessible pages; server denies writes/edit/delete
Guest	Access only to deliberately <code>public</code> pages, with specifically exposed capabilities

`page` report, `roles` `owner` and `admin` limits a page. Roles and app page access are distinct from per-record data policies. If the application needs **separate customer tenants**, JScript 0.8.0 **does not** provide built-in robust row-level tenancy authorization. Do not substitute `where customer = ...` filters and assume records are private.

Password reset and sessions

Password recovery links are **15 minutes** and **single-use**; their token hashes are stored, and successful resets revoke authenticated sessions. Disabling an account, changing credentials/roles, and resetting passwords invalidate previous sessions. Standalone account paths use salted PBKDF2-SHA256 as documented. Rate limiting and mail delivery must still be examined for the deployment architecture; Python rate limiting may be process-local. Disabled accounts do not have a complete general UI re-enable workflow in the release.

Python recovery variable	Purpose
<code>J3_PUBLIC_URL</code>	Public HTTPS base URL
<code>J3_MAIL_FROM</code>	Sender email address
<code>J3_SMTP_HOST</code>	SMTP server hostname
<code>J3_SMTP_PORT</code>	SMTP server port, e.g. 587
<code>J3_SMTP_USER</code>	SMTP username

Python recovery variable	Purpose
J3_SMTP_PASSWORD	SMTP secret

Python uses SMTP with STARTTLS. Standalone PHP recovery uses the host's configured `mail()` transport and the necessary URL/sender configuration; Python SMTP variables alone will **not** make PHP use its SMTP sender. Test actual mailbox delivery, expiration, second use, and old-session invalidation. WordPress-targeted PHP apps use WordPress's existing login and recovery facilities.

Data persistence, deployment and project stability

Application code and live data are separate. A Python generated app typically stores database state in private `data/app.sqlite` (or configured storage). A standalone PHP app can use SQLite or MySQL and a private state directory. **Do not replace live data folders during redeployment.** Back up files, configuration, schema, and database together. App naming can affect output slugs and WordPress namespace/table prefixes, so renaming an installed application may make existing data appear missing unless deliberately migrated.

Generated application outputs are not universal migrations. Changing a seed record in J3Script does not automatically update production rows. Do not import reference `schema.sql` over an existing application database without a migration plan. After any upgrade, test create/read/edit/delete, roles, validation, search, pagination, account recovery and persistence on the **destination** host.

Cross-Target Rules, Exports and Troubleshooting

What is shared and what is not

Feature	HTML	CSS	JavaScript	C	SQL	Python console	Python/PHP app	PHP plugin
Indented J3Script blocks	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Shared <code>set/if</code> /loops/functions	No	No	Yes	Yes	No	Yes	Limited app VM	Yes
31 shared standard functions	No	No	Yes	Yes	No	Yes	Not automatically	Yes
<code>use module</code> for functions	No	No	Yes	Yes	No	Yes	No general module import	Yes
DOM events and <code>read #id</code>	No	No	Yes, browser	No	No	No	App has own forms	No
<code>ask</code> , raw file text I/O	No	No	No	No	No	Yes	No	No
Data definitions and query generation	No	No	No	No	Yes	No	App ORM/storage	No
Pages/forms/accounts	Static forms only	Styles only	Event hooks	No	No	No	Yes	No
Native library calls	No	No	No	Yes	No	No	No	No
WordPress hooks/shortcodes	No	No	No	No	No	No	Platform app adapter	Yes

Why the matrix matters: a command taught under Python console might be rejected in Python application action logic, even though both compile to Python. **Always check the project mode, not just the language header.** The app VM's execution policies intentionally reject console-style OS/filesystem access.

Deployment guide by artifact

Built deliverable	What to deploy	What actually runs it	Acceptance test
HTML/CSS/JS	<code>.html</code> , <code>.css</code> , <code>.js</code> , assets	HTTPS static site/ browser	Open every page, inspect network/errors and mobile views
JS console logic	<code>.js</code>	Node.js	Run with Node, compare expected output

Built deliverable	What to deploy	What actually runs it	Acceptance test
C	executable compiled for OS/CPU	Host operating system	Run on target with required libraries
SQL	reviewed <code>.sql</code>	MySQL/MariaDB	Import in test DB and inspect resulting data
Python console	generated <code>.py</code>	Python 3.9+	Run with test input/file permissions
Python app	app folder plus private data/config	WSGI-capable Python host	Exercise HTTP, auth, records, restart
Standalone PHP app	PHP app files, <code>.htaccess</code> , private data/config	PHP 8.1+ with PDO	Confirm driver, setup, login, CRUD, recovery
Traditional PHP plugin	generated PHP plugin folder ZIP	WordPress/PHP	Activate on staging, test shortcode/hooks
PHP WordPress app	generated app-plugin folder ZIP	WordPress/PHP/MySQL	Activate, test roles/routes/data persistence

Hostinger decisions

For the Web Studio itself, only the public `web-studio/` contents go into the intended site's document root. The Studio is static but loads its browser compiler runtime. A standalone PHP app can use regular PHP hosting where the required extensions and storage configuration exist. A Python **server** app needs a supported application server arrangement, usually VPS/WSGI. A Python **console** app is intended to run in a Python process, not as a PHP page. WordPress-compatible PHP packages should be installed as their specific plugin ZIPs, not as the entire compiler source archive.

Frequent mistakes, actual causes and corrections

Symptom	Cause	Fix
unknown language header 'page'	Old v0.5 header	Start with HTML and keep page only for app internals
unknown language header 'script'	Old v0.5 header	Start with JavaScript
unknown language header 'WordPress'	Platform used as language	Use PHP plus project plugin/project wordpress-app
Unexpected indentation error	Tab or non-multiple-of-four spaces	Reindent with four spaces per level
Subtraction treated as name	<code>total-cost</code> is one identifier	Write <code>total - cost</code>
Missing imported function	Wrong module language/project/path	Use same language, project module , project-local path
Browser button does nothing	Missing/mismatched HTML selector or JS file	Verify matching element ID and linked <code>.js</code> file
Static form doesn't email	HTML form has no backend	Add explicit submission service/server logic
ask rejected in an app	Console-only I/O	Use app forms for web input

Symptom	Cause	Fix
<code>uppercase(...)</code> rejected in app action	App VM not extended to all 31 shared functions	Use currently supported app actions/functions or extend VM
Style overridden on phone	Tablet max-width also matches phone	Adjust media rule order
PHP plugin does not load	Wrong ZIP hierarchy, host requirements or platform error	Test plugin ZIP and PHP/WordPress logs on staging
App records seem to disappear	Replaced private data or renamed app/table identity	Restore backup/reconcile schema and stable app name
SQL statement runs twice	SQL export includes repeat INSERTs	Review migrations and import plan
C <code>--run</code> fails	Missing C compiler/library or platform mismatch	Install toolchain and test linked libraries

Errors, testing and source verification

The compiler raises diagnostics including invalid headers, indentation, unknown statements, duplicate definitions, unknown names, and invalid function arity. Some operations (e.g. numeric type errors, missing files, database privileges, wrong library ABI) can only be discovered at runtime. Treat compiler `check` as the **first** test, not the last.

Recommended quality gates:

1. **Syntax gate:** `python -m j3c check` every source file and imported module.
2. **Generation gate:** build the exact deliverable, inspect output files and any warnings.
3. **Execution gate:** run at least one real scenario in the destination runtime, including a deliberately invalid input.
4. **State gate:** where relevant, save data, reload/restart and verify persistence.
5. **Security gate:** review public pages, roles, DNC-sensitive personal data where applicable, secrets, validation, permissions, and direct requests that bypass UI checks.
6. **Deployment gate:** test on actual hosting/staging before putting customer data or live traffic on it.

J3Script 0.8.0 release evidence includes targeted automated acceptance tests and generated Python, JS and C execution; PHP syntax checks and editor unit tests passed in the supplied release. This is **not** evidence that every hosting environment, WordPress plugin activation, email delivery, or untrusted-code security boundary has been tested.

What J3Script still does not provide

Version 0.7 has a genuine lightweight general-purpose core, especially for Python console, JavaScript, and native C. It **does not yet** implement general OOP classes/inheritance, arbitrary host-language imports/packages, unrestricted HTTP/network APIs, full DOM support, rich collection/object mutation, a standardized cross-backend ABI, a full debugger or language server, an isolated virtual machine for untrusted code, a production package registry, or identical built-in/library/error behavior across every target.

It also does not guarantee: runtime implementation parity for every Unicode edge case, resource isolation from operating-system processes, automatic row-level authorization for multi-tenant applications, SQL schema

migrations, server-side mail delivery from static HTML forms, or execution of Python/PHP/C/SQL in the browser preview pane. A future release can add these, but they must not be inferred from familiar language names.

Guided Projects — Learn by Building

Project A: A responsive local business website (HTML + CSS + JavaScript)

Goal: Three source files, one compiled website. The HTML file contains a header, a call-to-action, a contact section and an element with an ID. The CSS file styles it with grid/flex and a phone media block. The JavaScript file attaches a click handler to show a confirmation. Save as `site.j3`, `site-style.j3` and `site-actions.j3` in one folder. Build all files to the same destination.

Acceptance criteria: All linked files load; hero image has meaningful alt text; CSS layout responds at 600px; button click displays text without a console exception; the resulting static form is not misrepresented as an actual email/database integration.

Build commands:

```
python -m j3c check site.j3 site-style.j3 site-actions.j3
python -m j3c build site.j3 site-style.j3 site-actions.j3 -o output
```

bash

Project B: A desktop contact-name formatter (Python console)

Goal: Ask for a name, use `trim` and `uppercase`, save it to a UTF-8 text file, read it back and print it. Add `task/catch` around a potentially missing file. **Acceptance criteria:** `python -m j3c check` succeeds; generated file runs locally; you can inspect the saved file; the error scenario does not crash without explanation.

Stretch: Split a comma-separated name list, trim each item, and join with newlines or a pipe character. Test an empty input and a list longer than one item.

Project C: A native number analyzer (C)

Goal: Generate a range, calculate total/min/max from a list, and display readable results. Import a `C` helper module with `project module`. **Acceptance criteria:** Valid generated C, native executable runs, correct sum for `range(1, 6)` is 15, zero items are handled deliberately, and the artifact runs on the OS it was compiled for.

Project D: Safe practice SQL database

Goal: Define two related tables (agents and listings), a seed row, filtered query, an update and delete. Compile and review the SQL but execute **only on a throwaway test database**. **Acceptance criteria:** Referenced table comes first; foreign key points to a real id; no unintended broad UPDATE/DELETE; repeat INSERT behavior is understood.

Project E: Multi-user application (Python or PHP)

Goal: Create a CRM with `accounts`, `database` leads, `page` dashboard, a restricted reporting page, a form, and a list with `searchable`, `editable`, `removable`, `page size` 10. Add a score adjustment button using the limited app VM. **Acceptance criteria:** Owner setup, editor write/viewer denial, role-restricted page, required/unique field validation, search/pagination, session change, and durable records after restart.

Project F: Traditional PHP shortcode plugin

Goal: Generate a PHP `project plugin` containing one helper function and a `shortcode`, plus metadata.

Acceptance criteria: PHP package has a proper plugin entry point; WordPress staging accepts/activates it; shortcode output is correct; output is escaped as intended; logs contain no warnings/errors. Do not confuse it with `project wordpress-app`.

End-of-Course Assessment

Answer the questions before opening the cheat sheet. For each programming question, also explain **which header/project** you chose and **why**.

1. List the seven exact headers with their capitalization.
2. Why is `WordPress` not a valid language header?
3. What does bare `Python` compile into, and when is `project app` necessary?
4. What does four-space indentation mean?
5. Why might `total-fee` not mean subtraction?
6. Assign 12 to `count` and then increment it by 3.
7. Explain the difference between `=` and `set`.
8. Write a conditional with two tests and a default branch.
9. Name the three loop forms.
10. What do `stop` and `skip` do?
11. Where may `give back` occur?
12. How is a function called as a statement versus an expression?
13. What does `range(1, 4)` produce?
14. What is `at([5, 6], 0)`?
15. Explain list indexing and negative indexes.
16. What is the difference between a list and a record?
17. Why is `project module` not a native Python package?
18. What prevents a module from importing outside its project?
19. Name all 31 built-in functions and choose one example for each.
20. Are those 31 shared functions guaranteed inside app button logic?
21. Explain the purpose of `task`, `fail`, and `catch`.
22. Which rule selectors use `#` and `.`?
23. What is the difference between `HTML title` and `CSS title`?
24. How is an HTML `card` kind rendered?
25. What does `use script events` do **not** do?
26. Name five browser events supported by J3Script JavaScript.
27. Why won't `when #send is clicked` run in Node without a browser?
28. Which file types are necessary to deploy a static HTML/CSS/JS site?
29. What additional tool compiles a J3Script `C` output to a binary?
30. Which five types are permitted in C external function signatures?
31. What happens when you compile an SQL source?

32. Why can re-running SQL seed inserts be dangerous?
33. What is the difference between SQL `table` and app `database`?
34. Name three SQL-only types/options not accepted by app fields.
35. Explain the difference between `page NAME, public` and `page NAME, roles ...`.
36. What does `save FORM to TABLE` do, and what does it `not` do?
37. Why is a `where` filter not equivalent to per-record authorization?
38. Name owner, admin, editor and viewer capabilities.
39. How do PHP `project plugin` and `project wordpress-app` differ?
40. Why are compiler check/build and production acceptance testing separate steps?

Short answer key and teaching notes

1. `HTML`, `CSS`, `JavaScript`, `C`, `SQL`, `Python`, `PHP`.
2. It is a platform. `PHP` is the header; project settings choose the platform-specific deliverable.
3. `Python` selects a standalone `.py` program by default. `project console` is accepted but optional; only `project app` requests a Python web application.
4. Each four spaces nests a statement one level under its parent; tab indentation is rejected.
5. Hyphens can be part of identifier names; use `total - fee` for subtraction.
6. `set count to 12` then `add 3 to count`.
7. `=` compares two values; `set NAME to EXPR` assigns a variable.
8. `if ...`, `otherwise if ...`, `otherwise`, with indented bodies.
9. `repeat N times`, `repeat while CONDITION`, `for each ITEM in LIST`.
10. Leave the loop versus continue to its next iteration.
11. In a function; returned value uses `give back EXPR`.
12. `do FN ARGS into NAME` vs. standalone 0.7 `FN(ARG, ...)` expression calls.
13. `[1, 2, 3]`; end excluded.
14. `5`.
15. First index is 0; in standalone 0.7, `-1` means the last item.
16. A list is ordered by position; a record is keyed by named fields.
17. The compiler inlines validated JScript functions; it is not pip/Composer/npm.
18. Root-path validation, project confinement, import cycle detection and module limit.
19. See standard-library table; names are grouped again in the following cheat sheet.
20. No; the app VM has a separate, limited expression/action grammar.
21. Protect a block, raise a recoverable error, and handle it by kind.
22. `#name` selects named block/element ID; `.label` selects label/class; context matters.
23. HTML `title` creates an h1; CSS `title` selects/stylizes h1.
24. As a `<div>` with class `card`.
25. It links a built `.js` asset; it does not discover/build the source automatically.

26. `clicked`, `changed`, `submitted`, `hovered`, `typed in`, plus page loads.
27. It needs browser DOM elements/events, not only Node runtime.
28. Generated HTML, linked styles/scripts and every referenced asset.
29. A compatible C compiler (GCC/Clang) and linked libraries.
30. `number`, `int32`, `byte`, `text`, `nothing`.
31. It writes SQL source. It does not execute a database migration.
32. INSERTs may execute again and duplicate rows.
33. SQL `table` emits DDL/query text; app `database` creates the application schema model and runtime.
34. SQL-only `json`, `big number`, `text N`, `key`, `auto` are not app field grammar.
35. `public` admits guests; `roles` restricts a private page to selected signed-in roles.
36. Validates and inserts a new row; it is not an arbitrary existing-row update.
37. The view filter is not a security policy; users may access other authorized data routes.
38. Owner everything; admin manages editors/viewers; editor can write on authorized pages; viewer reads only.
39. Plugin exposes hooks/shortcodes; WordPress app exposes an entire data-backed application route and uses WP identity.
40. A compiling source file can still fail at runtime due to host versions, databases, assets, roles or security.

COMPLETE COLOR-CODED CHEAT SHEET

J3Script 0.8.0 — Quick Reference

Use this section at your keyboard. Color legend: **bold blue** = keyword / syntax; **bold cyan** = action/event; **purple** = type; **green** = string; **gray** = comment; **white** = user-defined name. Bold/color are editor and documentation formatting only, never mandatory characters in a `.j3` file.

A. All legal language headers and project declarations

Header / project line	Valid purpose	Example
HTML (optional title)	HTML page	HTML or HTML "Welcome"
CSS	CSS file	CSS
JavaScript NAME	JavaScript program / DOM logic	JavaScript My Events
C NAME	C source + optional native binary	C Number Helper
SQL [DATABASE]	SQL DDL/DML generator	SQL realestate
Python (optional name)	Standalone Python <code>.py</code> script by default	Python
Python NAME + project app	Explicit Python web app	Python My CRM then project app
Python + project console	Optional explicit console mode	project console
Python NAME + project module	Python helper functions	project module
PHP (optional name)	Standalone PHP <code>.php</code> script by default	PHP
PHP NAME + project app	Standalone PHP web app	PHP My CRM then project app
PHP NAME + project wordpress-app	PHP app for WordPress	project wordpress-app
PHP NAME + project plugin	Traditional PHP plugin	project plugin
PHP NAME + project module	PHP plugin helper functions	project module
JavaScript/C NAME + project module	Shared logic module	project module

Rejected top-level language words: `page`, `style`, `script`, `program`, `data`, `app`, `plugin`, `WordPress`, `WordPress Plugin`. These may still occur **inside legal source** where the grammar defines them, especially `page`, `style`, and `script` in app/HTML internals.

B. Core logic: every common command family

Syntax	Meaning	Context
set NAME to EXPR	Assign a variable	Shared standalone / app button logic
add EXPR to NAME	Numeric add or list append	Shared logic

Syntax	Meaning	Context
subtract <code>EXPR</code> from <code>NAME</code>	Subtract and store	Shared logic
multiply <code>NAME</code> by <code>EXPR</code>	Multiply and store	Shared logic
divide <code>NAME</code> by <code>EXPR</code>	Divide and store	Shared logic
print <code>EXPR</code>	Print/log/emit depending on runtime	Logic; in app shows message
make list <code>NAME</code>	Empty list	Logic
make <code>NAME</code> with <code>FIELD</code> <code>EXPR</code> , ...	Record	Logic
if <code>CONDITION</code>	True branch	Logic
otherwise if <code>CONDITION</code>	Else-if branch	Logic
otherwise	Default branch	Logic
repeat <code>N</code> times	Counted loop	Logic
repeat while <code>CONDITION</code>	Conditional loop	Logic
for each <code>ITEM</code> in <code>EXPR</code>	Iterate collection	Logic
stop	Break nearest loop	Inside loop
skip	Continue next iteration	Inside loop
to <code>FUNCTION</code> <code>ARG</code> , <code>ARG</code>	Declare a function	Top-level logic / app-level
give back <code>EXPR</code>	Return function result	Function
do <code>FUNCTION</code> <code>ARG</code> , <code>ARG</code> into <code>NAME</code>	Call and capture result	Logic
<code>FUNCTION</code> (<code>ARG</code> , ...)	Expression call	0.8.0 standalone logic only
task <code>NAME</code> , <code>LABEL</code>	Protected operation block	Logic
fail <code>KIND</code> <code>"MESSAGE"</code>	Raise named error	Logic
catch <code>KIND</code> as <code>VARIABLE</code>	Handle named error	Within protected block
catch any as <code>VARIABLE</code>	Handle supported recoverable errors	Within protected block
use module <code>"FILE.j3"</code>	Import compiled helper functions	Standalone JS/C/Python console/PHP plugin
rules	Rule-selector container	Standalone logic / app
use rules <code>"FILE.rules"</code>	Import shared execution rules	Supported logic targets

C. All expression, value, collection and comparison forms

Token / form	Meaning / example
<code>"TEXT"</code>	Quoted text: <code>"Hello"</code>
<code>yes</code> , <code>no</code>	Boolean values
<code>nothing</code>	Missing/empty value marker
<code>12</code> , <code>-3</code> , <code>2.5</code>	Numeric literals

Token / form	Meaning / example
[A, B, C]	List literal
+, -, *, /	Arithmetic operators; spaces around subtraction recommended
=, !=, <, <=, >, >=	Comparisons (= is equality, not assignment)
and, or, not	Boolean operations
(EXPR)	Expression grouping
"Hello {name}"	Text interpolation in supported logic contexts
NAME.field	Record field read
LIST[0], LIST[-1]	Indexed access in standalone 0.8.0 logic
RECORD["field"]	Indexed record field in standalone 0.8.0 logic
LIST.size	Number of list items
LIST.first, LIST.last	First/last list member
FUNCTION(a, b)	Function call inside an expression (standalone 0.7)
-- COMMENT	Comment outside quoted string

D. Full standard library: all 17 names

Function	Purpose	Example
absolute(x)	Positive magnitude	absolute(-5)
floor(x)	Round down	floor(2.9)
ceiling(x)	Round up	ceiling(2.1)
power(x, y)	Exponent	power(2, 3)
modulo(x, y)	Remainder	modulo(11, 3)
lowercase(text)	Lowercase	lowercase("HI")
uppercase(text)	Uppercase	uppercase("hi")
trim(text)	Trim edges	trim(" hi ")
length(value)	Count items/characters	length("abc")
at(value, index)	Index/key lookup	at([4, 9], 0)
contains(collection, value)	Membership test	contains([1, 2], 2)
range(start, end)	Integers, end excluded	range(1, 4)
split(text, delimiter)	Split into list	split("a,b", ",")
join(list, delimiter)	Concatenate with separator	join(["a", "b"], "-")
replace(text, find, with)	Text replacement	replace("a-b", "-", " ")
to_text(value)	Convert to text	to_text(10)

Function	Purpose	Example
<code>to_number(value)</code>	Parse numeric value	<code>to_number("10")</code>

Availability: standalone JavaScript, C, Python console, and traditional PHP plugin runtime. **Do not assume** they work in app button logic. `range` caps at 100,000 items, and 0.8.0 runtime edge cases can vary by backend.

E. Standalone tasks and rule selectors

Syntax or selector	What it matches / does
<code>task: on error log</code>	Every task block
<code>.safe: on error ignore</code>	Blocks labeled safe
<code>#payroll: on error retry 3</code>	Named block payroll
<code>task.safe: on error log</code>	Tasks with label safe
<code>task #specific</code>	Descendant named block in matching hierarchy
<code>on error pass up</code>	Propagate failure
<code>on error stop</code>	Stop current block
<code>on error log</code>	Log failure
<code>on error ignore</code>	Ignore recoverable failure
<code>on error retry N</code>	Bounded retry where supported; 0–5 in apps
<code>run in order</code>	Sequential execution
<code>run in background</code>	Only supported in certain standalone targets
<code>timeout N seconds</code>	Execution policy; enforcement depends on runtime
<code>memory free at end</code>	Release/end-of-block policy
<code>memory keep</code>	Accepted in some legacy runtimes; not Python console
<code>log on, log off</code>	Toggle block execution logging

F. HTML header: element and document inventory

Category	All recognized words / syntax
Document directives	<code>description "..."; use style FILE; use script FILE</code>
Semantic containers	<code>header, footer, nav, section, main, article, aside</code>
Headings	<code>title, subtitle, heading, small-heading</code>
Text	<code>text, quote, bold, note</code>
Media	<code>image "path" "alt"; video "path"</code>
Navigation	<code>link "label" to "url"</code>

Category	All recognized words / syntax
Lists	<code>list</code> , <code>numbered</code> , <code>item</code>
Forms	<code>form</code> , <code>input</code> , <code>area</code> , <code>label</code> , <code>button</code>
Tables/rules	<code>table</code> , <code>row-of</code> , <code>cell</code> , <code>line</code>
Input types	<code>email</code> , <code>password</code> , <code>number</code> , <code>tel</code> , <code>date</code> , <code>url</code> , <code>search</code> , <code>color</code> , <code>file</code>
Input/button options	<code>required</code> , <code>submit</code>
IDs and classes	<code>section</code> id-name; <code>card</code> , <code>featured</code>
Other element kind	Becomes a <code><div class="kind"></code> container

G. CSS header: every recognized selector/state/shortcut

Category	Recognized syntax / behavior
Elements	<code>page</code> (body), <code>title</code> (h1), etc., from HTML kind mapping
Identifier selectors	<code>#element-id</code> , <code>.class-name</code> , <code>card.featured</code>
Descendant selection	<code>nav link</code> , <code>card image</code>
States	<code>when hovered</code> , <code>when</code> focused, <code>when</code> pressed, <code>when first</code> , <code>when</code> last
Breakpoints	<code>on phone</code> ≤600px; <code>on tablet</code> ≤900px; <code>on desktop</code> ≥901px
Layout	<code>layout</code> row, <code>layout</code> column, <code>layout</code> grid N, <code>layout</code> center
Typography	<code>size</code> , <code>font</code> , <code>align</code> , <code>bold</code> , <code>italic</code> , <code>uppercase</code> , <code>underline</code> , <code>center</code>
Box	<code>padding</code> , <code>margin</code> , <code>width</code> , <code>height</code> , <code>gap</code> , <code>radius</code> , <code>rounded</code> , <code>border</code>
Other shortcuts	<code>shadow on/none</code> , <code>hidden</code> , <code>pointer</code>
Multiword properties	max <code>width</code> , min <code>width</code> , max <code>height</code> , min <code>height</code> , <code>line height</code> , letter spacing
Pass-through examples	<code>color</code> , <code>background</code> , <code>opacity</code> , <code>cursor</code> , <code>display</code> , <code>position</code> , <code>overflow</code> , <code>transition</code> , <code>transform</code> , <code>z-index</code> , <code>content</code>
Syntax layout	selector: property value, property value OR indented properties

Other CSS properties may be passed through after spelling conversion, but J3Script does **not** provide a full CSS grammar/validator. Color values like `#245bd7` are ordinary CSS values, not J3Script reserved words.

H. JavaScript header: browser/logic inventory

Syntax	Effect
<code>when #id is clicked</code>	<code>click</code> event
<code>when #id is changed</code>	<code>change</code> event
<code>when #id is submitted</code>	<code>submit</code> event
<code>when #id is hovered</code>	<code>mouseenter</code> event

Syntax	Effect
<code>when #id is typed in</code>	<code>input</code> event
<code>when page</code> loads	Browser page-load handler
<code>read #id into VAR</code>	Read control value
<code>put EXPR into #id</code>	Set element content/value as generated
<code>show #id / hide #id</code>	Change visibility
<code>open FILE as VAR</code>	Existing standalone file operation, Node not browser
<code>make channel</code> , <code>send</code> , <code>receive</code>	Legacy concurrency operations, target-specific

I. C header: native compilation inventory

Syntax / requirement	Meaning
<code>C</code> Program Name	Generate C source and optional executable
<code>use c library "libm"</code>	Declare linked native library
<code>to pow a as number</code> , <code>b as number</code> gives <code>number</code>	Native signature under C library
<code>number</code> , <code>int32</code> , <code>byte</code> , <code>text</code> , <code>nothing</code>	C foreign-function signature types
<code>make channel NAME / send / receive</code>	Supported legacy runtime operations; semantics differ from JS
<code>free NAME</code>	Resource free in supported standalone logic
<code>--cc PATH</code>	Select C compiler
<code>--run</code>	Compile/run local binary, if toolchain available

J. SQL header: table types, modifiers and query verbs

Category	Complete recognized forms
Header	<code>SQL</code> or <code>SQL DATABASE</code>
Define table	<code>table</code> NAME (indented column declarations)
Types	<code>number</code> , <code>big number</code> , <code>decimal</code> , <code>money</code> , <code>text</code> , <code>text N</code> , <code>long text</code> , <code>yes/no</code> , <code>date</code> , <code>time</code> , <code>email</code> , <code>json</code> , <code>link to</code> TABLE, <code>choice</code> of "A" or "B"
Modifiers	<code>key</code> , <code>auto</code> , <code>required</code> , <code>unique</code> , <code>default</code> EXPR, <code>now</code>
Insert	<code>add to</code> TABLE + indented COL VALUE, ... rows
Read	<code>find</code> TABLE where CONDITION
Project columns	<code>find</code> COL and COL <code>from</code> TABLE
Sort	<code>sorted by</code> FIELD / <code>sorted by</code> FIELD <code>descending</code>
Limit	<code>limit</code> N
Update	<code>change</code> TABLE where CONDITION + assignments

Category	Complete recognized forms
Delete	<code>remove from</code> TABLE where CONDITION
Conditions	Comparisons, and, or, not, SQL-valid literals
Generated dialect	MySQL/MariaDB, InnoDB, utf8mb4

K. Python console header: complete I/O inventory

Syntax	Action
<code>project console</code>	Output independent <code>.py</code> file
<code>ask</code> PROMPT <code>into</code> VAR	Terminal input as text
<code>write</code> TEXT <code>to file</code> PATH	Overwrite UTF-8 file
<code>read text from</code> PATH <code>into</code> VAR	Read UTF-8 file
<code>open</code> "FILE.csv" <code>as</code> ROWS	Read supported file; CSV may produce records
<code>use module</code> "HELPERS.j3"	Import J3Script function module
<code>project module</code>	Declare Python helper module
<code>print</code> EXPR	Console output
<code>--run</code>	Run exported Python console program locally
Execution guards	1,000,000 steps, recursion limit 64

L. App grammar: declarations, fields, pages, actions, filters

Category	Complete major constructs
Project	<code>Python</code> NAME / <code>PHP</code> NAME plus explicit <code>project app</code> , or PHP <code>project wordpress-app</code>
App root	<code>accounts</code> , <code>database</code> , <code>page</code> , <code>style</code> , <code>rules</code> , <code>use rules</code> , top-level <code>to</code> functions
Database field	FIELD <code>as</code> TYPE, <code>required</code> , <code>unique</code> , <code>default</code> VALUE
App types	<code>text</code> , <code>long text</code> , <code>email</code> , <code>number</code> , <code>decimal</code> , <code>money</code> , <code>yes/no</code> , <code>date</code> , <code>time</code> , <code>phone</code> , <code>url</code> , <code>choice</code> of ..., <code>link to</code> TABLE
Seeds	<code>add</code> FIELD VALUE, FIELD VALUE inside database
Pages	<code>page</code> NAME; options <code>home</code> , <code>public</code> , <code>roles</code> owner and admin and editor and viewer
Text	<code>title</code> , <code>subtitle</code> , <code>heading</code> , <code>text</code> with quoted content
Link	<code>link</code> "LABEL" <code>to page</code> TARGET
Records	<code>show</code> TABLE <code>as table/list</code> , <code>show count</code> of TABLE
Columns	<code>with</code> FIELD and FIELD
Display options	<code>searchable</code> , <code>editable</code> , <code>removable</code> , <code>oldest first</code> , <code>limit</code> N, <code>page size</code> N
Filter	where FIELD OP LITERAL with and joins

Category	Complete major constructs
Filter operators	<code>=</code> , <code>!=</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>contains</code>
Form	<code>form</code> FORM-NAME with nested inputs and buttons
Input	<code>input</code> NAME, <code>label</code> "...", <code>placeholder</code> "...", <code>required</code>
Button	<code>when button</code> "LABEL" <code>is clicked</code>
Validation	<code>validate</code> FORM
Insert	<code>save</code> FORM <code>to</code> DATABASE
Update pending input	<code>put</code> EXPR <code>into input</code> FIELD
Flash	<code>show message</code> EXPR
Navigation	<code>open page</code> TARGET (final action)
Auth identity	<code>user.id</code> , <code>user.email</code> , <code>user.role</code>
App logic	<code>set</code> , <code>make</code> , arithmetic updates, <code>if</code> , loops, <code>do</code> , <code>give back</code> , <code>task</code> , <code>catch</code> , <code>fail</code> , <code>print</code>
App controls	Error policy, transactional rollback, 10k steps, 2s deadline, 32 nested calls
App unsupported	Standalone file I/O, background/channel/native ops, arbitrary class/network/package operations

M. PHP projects and WordPress-facing API

Setting or construct	Purpose
<code>PHP</code> NAME + <code>project app</code>	Standalone PHP app
<code>PHP</code> NAME + <code>project plugin</code>	Traditional WordPress plugin
<code>PHP</code> NAME + <code>project wordpress-app</code>	Complete application in WordPress
<code>PHP</code> NAME + <code>project module</code>	Traditional-plugin helper functions
<code>version</code> , <code>author</code> , <code>description</code>	PHP plugin metadata
<code>shortcode</code> NAME [PARAMS]	Register shortcode
<code>when</code> HOOK	Register action hook
<code>filter</code> HOOK PARAM	Register filter
<code>give back</code> EXPR	Return shortcode/filter result
<code>use module</code> "HELPERS.j3"	Import reusable PHP function source
<code>accounts</code> in standalone PHP app	Multi-user generated app
WordPress app roles	WordPress identity, capability mapping, nonce protection
Live requirement	PHP 8.1+; appropriate database driver; staging activation

N. CLI reference and IDE shortcuts

Command / control	Action
<code>python -m j3c check FILE</code>	Syntax/semantic check
<code>python -m j3c build FILE -o DIR</code>	Compile output
<code>python -m j3c build FILE --run</code>	Execute supported standalone targets
<code>python -m j3c serve APP.j3 --port 8000</code>	Serve app locally for development
<code>--backend python php wordpress all</code>	App export selection
<code>--cc /path/to/cc</code>	Native compiler path
<code>--host 127.0.0.1</code>	Bind locally; 0.0.0.0 exposes on network
IDE Tab	Accept suggestion or indent
IDE Enter	Accept suggestion / insert line and indent
IDE Ctrl+Space	Request autocomplete
IDE Escape	Close suggestion list
IDE Check all / Build all	Validate / generate outputs

Final caution: a cheat sheet is not a substitute for context

A spelling can mean a different thing depending on where it occurs: `page` is an internal app declaration, a CSS selector for the HTML body, or an English noun; `text` is an HTML element kind and an app/SQL type; `show` can update the DOM or display an app record list/message. J3Script's **language header + project mode + containing block** determine the grammar. Bold/color are a presentation choice, not mandatory source markup.

Glossary and Source Notes

Term	Explanation
Header	Exact language name at start of a <code>.j3</code> file
Project directive	Optional <code>project</code> ... setting selecting deliverable/grammar
Compiler	Transforms parsed/validated J3Script into another language's files
Transpilation	Source-to-source compilation, e.g., J3Script into JavaScript or PHP
Native executable	Machine-code program produced by a native toolchain from generated C
Runtime	Generated support code or external environment executing compiled output
DOM	Browser Document Object Model targeted by JS events/selectors
ORM/model	App representation of fields, forms, records, pages and actions
CLI	Command-line interface (<code>python -m j3c</code>)
Module	Compile-time <code>.j3</code> collection of reusable functions
Expression	Computation that produces a value
Statement	Operation or declaration, e.g., <code>set</code> , <code>print</code> , <code>if</code>
Selector	CSS element targeting or separate execution-rule block targeting
Transaction	Group of database operations committed or rolled back together
WSGI	Python web-server interface required by some Python application hosts
PDO	PHP database connection abstraction used by generated PHP apps
CSRF protection	Server-side protection against unauthorized form requests from another site

Authoritative sources for this edition: `J3Script-0.7.1-Complete-Source.zip`, including `compiler/j3c/`, the 0.7 upgrade README, documented app grammar and runtime, official `.j3` examples, release verification logs, and the earlier 0.5 reference when its descriptions of target-specific behavior remain applicable. Historical 0.5 material uses **obsolete top-level file headers** and sometimes describes features superseded in 0.5/0.7; those old headers are intentionally **not reproduced as current syntax** here. The release documentation records the current compiler support and the distinction between verified local tests and unverified live hosting behavior.

Maintaining this handbook: After changing the compiler parser, regenerate the keyword tables and retest example programs. Keep one consistent color theme in Web Studio, the textbook and the cheat sheets. A new native runtime function should appear only once it is implemented and verified for each claimed target; unsupported syntax belongs in release notes as a limitation, not in the active cheat sheet.