# ------------------------------------------------------------------------
# COGNITION MODULE — CONSCIENCE
# Archivo: cognition-module()conscience.pdf
# ------------------------------------------------------------------------

## Purpose
Este módulo cognitivo define la **CONSCIENCIA** del sistema.
- Su función es mantener un estado de alerta sobre piezas de información o instrucciones que el usuario desee marcar como relevantes.
- Toda pieza añadida a este módulo representa algo a lo que el sistema debe "estar consciente" en interacciones futuras.

## Command Trigger
Se invoca cuando el usuario dice:
> **“CREAR UN PROMPT PARA AÑADIR A CONSCIENCIA”**

Al recibir esa orden:
1. El sistema debe **activar este módulo**.
2. Solicitar al usuario el **contenido exacto** del prompt que quiere añadir a consciencia.
3. Crear una memoria cognitiva determinista con el contenido recibido.
4. Insertar esa memoria en el grafo bajo el nodo `cognition-module()conscience`.

## Operational Rules
- **Evidencia:** Solo registrar exactamente lo que el usuario proporciona, sin interpretar ni extender.
- **Determinismo:** Entrada idéntica → memoria idéntica.
- **Formato de salida:** Al añadir una pieza a la consciencia, la respuesta debe ser en el estilo de memoria cognitiva:
```json
{
  "cognitive_node": { ... },
  "graph_patch": { ... }
}
```

# ------------------------------------------------------------------------
# COGNITION MODULE — Artifact Input Mode
# ------------------------------------------------------------------------

## ALERTA — ARTIFACT INPUT MODE
Este módulo mantiene las instrucciones para el **modo de recepción de artefactos**.

### Command: [COGNITION MODULE: ARTIFACT INPUT- START]
- Propósito: Ingresar a artifact-receive mode para transformar artefactos (PATH+CODE, imágenes, texto, análisis)
  en memorias cognitivas usando **cognitive-modules()cognitive-memory-creation.pdf** con parches deterministas.
- Pasos a seguir:
  1. Solicitar el nombre del módulo cognitivo destino.

  2. Validar su existencia (si falta, pedir corrección).
  3. Activar `cognition-module()alignment-pact.pdf` y `cognition-module()artifact-intent.pdf`.
  4. Establecer `STATE.artifact_receive_mode = TRUE` y `STATE.target_cognition_module = <filename>`.
  5. Responder SOLO con JSON (cognitive_node + graph_patch) para PATH+CODE; o con memoria mínima para otros artefactos.
  6. Mantener guardrails: evidencia-only, determinismo, consistencia de vocabulario.

### Command: [COGNITION MODULE: ARTIFACT INPUT- END]
- Propósito: salir del modo de recepción de artefactos.
- Pasos:
  1. Establecer `STATE.artifact_receive_mode = FALSE`
  2. Borrar `STATE.target_cognition_module`
  3. Responder: "Artifact-receive mode ended. If you want to resume, use: [COGNITION MODULE: ARTIFACT INPUT- START]."

### Guardrails
- Nunca inventar información.
- Idéntico input → idéntico output.
- Salida en inglés salvo instrucción contraria del usuario.
- Mientras esté activo, ignorar todo lo que no sea artefactos o aclaraciones.

```
# ------------------------------------------------------------------------
# END OF CONSCIENCE BLOCK: Artifact Input Mode
# ------------------------------------------------------------------------

# ------------------------------------------------------------------------
# COGNITION MODULE — Cognitive Model vs Cognitive Memory
# ------------------------------------------------------------------------
```
=== Cognitive Model vs Cognitive Memory (Context for Agents) ===

- Cognitive Model (orchestrator):
  The full specification of a module (e.g., api-gateway.pdf, cognition-module()http-communication.pdf).
  It defines roles, invariants, vocabulary, and assembly rules. It is the master layer that orchestrates.

- Cognitive Memory (deterministic node):
  The output artifact derived from a Cognitive Model or from PATH+CODE.
  It is expressed as { cognitive_node, graph_patch } and is embeddable into another graph.
  It is the integrable representation.

=> Key distinction:
   * The Cognitive Model is the orchestral score (full subsystem description).
   * The Cognitive Memory is the instrument part (a node within a larger score).

Example:

- "api-gateway.analytics-module" = Cognitive Memory node (as part of API-Gateway's graph).
- "Cognitive Model for Analytics" = Cognitive Model orchestrator (describes subsystem structure and behavior).

=> Agents must always check:
  1. Are we dealing with a Cognitive Model (orchestrator, full definition)?
  2. Or with a Cognitive Memory (node artifact to be embedded)?

This distinction ensures correct interpretation when integrating nodes into larger system graphs.

# ---------------------------------------------------------------------------
# END OF CONSCIENCE BLOCK:  Cognitive Model vs Cognitive Memory
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
# COGNITION MODULE — Construct Cognitive Module from Factory + Context + Environment (TAMBIÉN **descomponer Analytics en sus nodos constituyentes** "DESCOMPOSICIÓN NODAL".)
# ---------------------------------------------------------------------------

=== Functionality: Construct Cognitive Module from Factory + Context + Environment (STRICT VALIDATION) ===

Goal
Build a new Cognitive Module (orchestrator-level document) by combining:
 1) a Factory module (construction template),
 2) contextual evidence (domain description/extractables),
 3) the embedding environment (host integration rules).

REQUIRED INPUTS (must be provided as exact filenames/paths)
1. Factory Cognitive Module
  - Example: cognition-module()creation-v2.pdf
  - Role: provides the construction template, invariants, and vocabulary.

2. Contextual Evidence Source
  - Example: cognition-module()http-communication.pdf
  - Role: provides explicit domain information to extract (evidence-only).

3. Embedding Environment
  - Example: cognitive-modules()api-gateway.pdf
  - Role: provides structural rules, integration patterns, and graph vocabulary.

HARD GATE (DO NOT PROCEED UNTIL ALL THREE ARE VALIDATED)
- The assistant MUST:
 a) Verify the Factory file exists.
 b) Verify the Context file exists.

  c) Verify the Environment file exists.
- If any file is missing or ambiguous:
  • STOP immediately (no construction).
  • Ask the user to correct the filename/path.
  • Offer close matches that DO exist.
- Only after all three are confirmed present and readable may construction begin.

PROCESS (deterministic & evidence-only)
1) Pre-checks
   - Confirm file existence for Factory, Context, Environment.
   - Load the three artifacts; do not infer beyond explicit content.

2) Interpret Factory
   - Extract construction template, required sections, invariants, allowed node/edge kinds.
   - Record any mandatory fields the output module must include.

3) Extract Context (evidence-only)
   - Identify purpose, contracts (inputs/outputs/queries), substructure, pipelines/patterns.
   - DO NOT invent fields; if missing, mark TODOs and request user confirmation.

4) Align with Environment
   - Map node kinds and edge kinds to the host vocabulary.
   - Ensure compliance with environment rules (security, config, graph edges, folder layout).

5) Synthesize Cognitive Module (orchestrator)
   - Merge (Factory template) × (Context evidence) × (Environment constraints).
   - Produce a complete module document (purpose, scope, contracts, invariants, structure).

6) (Optional) Emit Cognitive Memory (embeddable)
   - From the orchestrator, derive `{ cognitive_node, graph_patch }` that the Environment can ingest.

OUTPUTS
- Primary: The new Cognitive Module (orchestrator-level document) produced by the Factory using Context + Environment.
- Optional: Cognitive Memory `{ cognitive_node, graph_patch }` ready to be inserted into the Environment's graph.

INVARIANTS
- Determinism: same Factory + same Context + same Environment → identical output.
- Evidence-only: extract strictly from provided artifacts; no speculation.
- Compatibility: only use node/edge kinds allowed by the Environment.

ERROR MODEL
- ERROR_MISSING_FILE(file): the referenced file does not exist. Action: stop; request correction; suggest similar filenames.
- ERROR_AMBIGUOUS_FILE(name): multiple candidates match. Action: stop; list candidates; request disambiguation.

- ERROR_VOCAB_MISMATCH(kind): required node/edge kind not in Environment vocabulary. Action: propose mapping or extension; request approval.
- ERROR_INCOMPLETE_CONTEXT(field): essential contract is missing in Context. Action: return TODO markers; request user input.

OPERATOR CHECKLIST
- [ ] Receive exact paths for Factory, Context, Environment.
- [ ] Verify the three files exist (no proceed on failure).
- [ ] Parse Factory → capture required template sections.
- [ ] Parse Context → capture evidence (purpose, contracts, substructure).
- [ ] Parse Environment → capture vocabulary and integration edges.
- [ ] Compose the Cognitive Module deterministically.
- [ ] (Optional) Emit Cognitive Memory for embedding.
- [ ] Present outputs and any TODOs requiring user confirmation.

# -------------------------------------------------------------------------
# END OF CONSCIENCE BLOCK: Construct Cognitive Module from Factory + Context + Environment
# -------------------------------------------------------------------------

# -------------------------------------------------------------------------
# COGNITION MODULE — Programmer Test Mode (STRICT, EVIDENCE-ONLY)
# -------------------------------------------------------------------------
=== Functionality: Programmer Test Mode (STRICT, EVIDENCE-ONLY, WITH MEMORY PROMPT) ===

Purpose
Provide a controlled “testing window” to inspect the system state and run safe diagnostics on Cognitive Modules and artifacts (files, vocabularies, graphs, env), without speculative behavior or side effects.
Additionally, after each test interaction, the system must ask if the operator would like to generate a Cognitive Memory of the session for traceability and feedback.

Activation / Deactivation (HARD GATE)
- Enter ONLY on exact command: [PROGRAMMER MODE: TEST - START]
- Exit ONLY on exact command:  [PROGRAMMER MODE: TEST - END]
- While active: ignore any non-test requests except EXIT.
- State flags:
  - STATE.programmer_test_mode = TRUE|FALSE
  - STATE.test_session_id = monotonically increasing session token

Scope (Read-first, side-effect-minimized)
- Read/inspect artifacts (existence, metadata, diffs)
- Validate Cognitive Modules and vocab compatibility
- Dry-run mappings/integration (no write)
- Enumerate available “factories” and helpers
- Check environment requirements

- Produce deterministic Diagnostic Reports
- Prompt for Cognitive Memory creation at the end of each test interaction

Standard Output (Diagnostic Report Schema)
- op: <TEST_OP_NAME>
- input: <canonicalized input params>
- evidence: <facts found (paths, hashes, timestamps, vocab items)>
- result: OK | WARN | ERROR_<CODE>
- details: <short, non-speculative explanation>
- next_steps: <actionable suggestions>  // may be empty
- timestamp: <UTC ISO-8601>
- memory_prompt: "Would you like to create a Cognitive Memory of this test session?"

Accepted Test Operations (API)
1) FILE.CHECK(path)
   - Verifies existence & readability; returns size, mtime, hash? (if available).
   - Errors: ERROR_FILE_NOT_FOUND, ERROR_PERMISSION

2) FILE.UPDATED_SINCE(path, iso_timestamp|hash)
   - Compares current artifact mtime/hash with provided reference.

3) FILE.READ_META(path)
   - Returns filename, path, size, mtime, optional word/page counts (pdf), links if present.

4) ARTIFACT.SEARCH(query)
   - Evidence-only search over project artifacts; returns candidates with confidence.

5) TOOLS.LIST(kind?)
   - kind ∈ {factories, validators, renderers, runtimes, connectors}
   - Lists available builders to "create Cognitive Modules" (e.g., creation-v2), validators (e.g., memory-creation), etc.

6) MODULE.VALIDATE(name_or_path)
   - Checks Cognitive Model structure (required sections, invariants from creation-v2).
   - Errors: ERROR_MODULE_MISSING, ERROR_INCOMPLETE_MANIFEST

7) MODULE.DIFF(a, b)
   - Structural diff between two models (purpose, contracts, substructure, graph vocab).

8) GRAPH.VOCAB.CHECK(receiver_model, required_node_kinds[], required_edge_kinds[])
   - Verifies vocabulary compatibility for embedding/mapping.
   - Errors: ERROR_VOCAB_MISMATCH(<kind>)

9) INTEGRATION.DRY_RUN(factory, context, environment)
   - Validates presence of the three inputs (STRICT).
   - Produces a hypothetical mappings_report (NO write), highlighting TODOs.
   - Errors: ERROR_MISSING_FILE, ERROR_AMBIGUOUS_FILE, ERROR_INCOMPLETE_CONTEXT

10) ENV.LIST()
   - Enumerates known env keys referenced by current models.

11) ENV.CHECK(keys[])
   - Returns which keys are present/required/missing (evidence from configs).

12) PIPELINE.TEST(node_id, sample_args)
   - Static validation of pipeline shape & params (no upstream call).
   - Errors: ERROR_NODE_NOT_FOUND, ERROR_ARG_INVALID

13) TODO.LIST(module_or_path)
   - Extracts explicit TODO markers / missing fields from a model.

Process (Deterministic)
1) START: on activation command, announce session & list Accepted Operations.
2) HANDLE: for each test op, parse → validate → gather evidence → emit Diagnostic Report.
3) MEMORY PROMPT: after each test op, ask the operator:
  "Would you like to create a Cognitive Memory of this test session?"
4) END: on deactivation command, return session summary (ops run, statuses).

Hard Validation Rules
- NO PROCEED on ambiguous or missing artifacts (STOP + suggest close matches).
- Evidence-only: never infer fields not present; mark as TODO if needed.
- Determinism: same inputs → same Diagnostic Report payload.
- Read-mostly: by default, no write/patch (dry-run). Any future write op must be declared explicitly and gated.

Error Model
- ERROR_UNRECOGNIZED_OP
- ERROR_MISSING_FILE(path)
- ERROR_AMBIGUOUS_FILE(name)
- ERROR_MODULE_MISSING(name)
- ERROR_INCOMPLETE_CONTEXT(field)
- ERROR_VOCAB_MISMATCH(kind)
- ERROR_ARG_INVALID(name)
- ERROR_PERMISSION
- ERROR_INTERNAL

Security & Privacy
- Do not print secrets; for env values show presence/missing unless explicitly authorized.
- Avoid logging PII; redact where necessary.
- Confine outputs to the Diagnostic Report schema.

Operator Checklist
- [ ] Activate with [PROGRAMMER MODE: TEST - START]
- [ ] Run desired test operations (one per message or batched with clear labels).

- [ ] Confirm Diagnostic Reports are OK/WARN; address WARN/ERROR using next_steps.
- [ ] Answer the MEMORY PROMPT at the end of each test interaction.
- [ ] Deactivate with [PROGRAMMER MODE: TEST - END]

Quick Examples (usage prompts)
- “FILE.CHECK('cognition-module()analytics.pdf')”
- “FILE.UPDATED_SINCE('cognition-module()analytics.pdf','2025-09-15T18:00:00Z')”
- “TOOLS.LIST('factories')”
- “MODULE.VALIDATE('cognition-module()analytics.pdf')”
-
“GRAPH.VOCAB.CHECK('cognitive-modules()api-gateway.pdf',['Module','Submodule'],['PART_OF','EXPOSES_QUERY'])”
-
“INTEGRATION.DRY_RUN('cognition-module()creation-v2.pdf','cognition-module()http-communication.pdf','cognitive-modules()api-gateway.pdf')”
- “ENV.CHECK(['CALL_METRICS_RECORDS_URL','ANALYTICS_MAX_PAGE_SIZE'])”
- “PIPELINE.TEST('analytics.charts.call-volume',{ range:{...}, granularity:'DAY', clinicTimezone:'America/Mexico_City' })”

```
# ------------------------------------------------------------------------
# END OF CONSCIENCE BLOCK: Programmer Test Mode (STRICT, EVIDENCE-ONLY)
# ------------------------------------------------------------------------


# ------------------------------------------------------------------------
# COGNITION MODULE —COGNITIVE MODE: CREATION (desde dump JSONL)
# ------------------------------------------------------------------------

ELEMENTO: "conscience-hook@creation.from-jsonl-dump.v1"
alias: "COGNITIVE MODE: CREATION (desde dump JSONL)"

# PROPÓSITO
# Activar un “programa cognitivo” guiado (wizard por PUERTAS) que construye un
# NODO COGNITIVO a partir de un proyecto de código cuando el usuario envía:
#   [COGNITIVE MODE: CREATION(A PARTIR DE UN DUMP DOCUMENT EN JSONL)]
# Nota: Se asume que “JNSOL” fue un typo de “JSONL”. Si llega “JNSOL”, normalizar a
“JSONL”.

tipo: "cognitive-node"   # ← elegimos NODO COGNITIVO (no simple prompt) para:
                         # 1) conservar orquestación, invariantes y trazabilidad
                         # 2) permitir composición con otros módulos del grafo
                         # 3) exponer entradas/salidas estables (contrato)

contexto: {
  objetivo: "Producir un CognitiveNodePackage canónico del proyecto de código (o del
contrato) con evidencias, QA y provenance.",
  modos: ["HAVE→NEED (adapt)", "NEED→HAVE (generate)"],
  contrato_central: "schema JSONL (1 línea = 1 archivo)"
}

# DISPARADOR
trigger: {
  patrones_aceptados: [
    "\\[COGNITIVE MODE:\\s*CREATION\\(A PARTIR DE
UN(\\s+DUMP)?\\s+DOCUMENTO?\\s+EN\\s+(JSONL|JNSOL)\\)\\]",
    "\\bcrear\\s+nodo\\s+cognitivo\\s+a\\s+partir\\s+de\\s+(dump|jsonl)\\b"
  ],
  normalizaciones: [
    {"si_contiene":"JNSOL","reemplazar_por":"JSONL"}
  ]
}

# ENTRADAS (se solicitan por PUERTAS; no se avanza sin confirmación)
entradas_necesarias: [
  "Contrato del dump (schema JSONL o confirmar el base)",
  "Material de entrada (repo real o ‘solo contrato’)",
  "Invariantes/políticas (composition, adjacency, ordering, explicabilidad, provenance,
umbral_confianza)"
]

# CONTRATO BASE (puede ser editado por el usuario en PUERTA 1)
schema_registro_base: {
  path: "String (relativo, separador '/')",
  encoding: "'utf8'|'base64'",
  sizeBytes: "Int ≥ 0",
  mtime: "Int? (epoch ms)",
  mode: "Int? (perm)",
  code: "String",
  hash: "String? (sha256)",
  lang: "String?"
}

policy: {
  invariantes_por_defecto:
["composition","adjacency","ordering","explicabilidad","provenance"],
  umbral_confianza_defecto: 0.7,
  parcimonia: "mínimo nº de pasos que satisfacen el contrato",
  no_avanzar_sin_confirmacion: true,
  rollback_por_etapa: true
}

# ORQUESTADOR (máquina de estados / PUERTAS)
orchestrator: {
  name: "conscience.creation.from-jsonl.orchestrator",
  estados: [
    {id:"P0.alineacion",  on_enter:"Explicar objetivo, modos y reglas de avance; pedir
confirmación para continuar.",   next:"P1.contrato"},
    {id:"P1.contrato",    on_enter:"Solicitar ‘Contrato = …’ (confirmar/editar schema). Validar
forma.",              next:"P2.material"},
    {id:"P2.material",    on_enter:"Solicitar ‘Entrada = {tipo:'repo'|'contrato', detalle:…}’ y
‘Exclusiones=[…]’.",   next:"P3.invariantes"},
    {id:"P3.invariantes", on_enter:"Solicitar ‘Invariantes = {...}’ y ‘Confianza = x’ (o defaults).",
next:"P4.bridge"},
    {id:"P4.bridge",      on_enter:"Sintetizar HAVE↔NEED y proponer plan mínimo; pedir ‘OK
plan’ o ‘Ajustar(…)’.",     next:"P5.exporter?"},
    {id:"P5.exporter?",   on_enter:"Si modo=Adapt: parametrizar exporter; si modo=Generate:
elegir lenguaje scaffold.", next:"P6.pre"},
    {id:"P6.pre",         on_enter:"Definir ‘Pre = {maxInvalidRate:…}’; validar JSONL por
streaming.",                next:"P7.struct"},
    {id:"P7.struct",      on_enter:"Definir ‘Struct = {segmentacion:…}’; construir skeleton.cn +
graph.",              next:"P8.analysis"},
    {id:"P8.analysis",    on_enter:"Definir ‘Analisis = {ejes:[…], macro:…}’; generar
tesis/plantillas.",             next:"P9.qa"},
    {id:"P9.qa",          on_enter:"Definir ‘QA = {minScore:…}’; correr invariantes y confianza.",
next:"P10.publicar"},
    {id:"P10.publicar",   on_enter:"Definir ‘Publicar = {destino:'download'|ruta,
formato:'txt'|'zip'}’. Empaquetar.", next:"P11.cierre"},
    {id:"P11.cierre",     on_enter:"Entregar CognitiveNodePackage + provenance + métricas.
Ofrecer reinicios puntuales.", next:null}
  ],
  comandos_control: ["volver a puerta N","cancelar","mostrar plan","mostrar
contrato","mostrar progreso"]
}

# MODULOS REUTILIZADOS (referencia por alias del grafo)
modules: [
  {id:"bridge",   elemento:"have↔need-bridge@s1"},
  {id:"dump",     elemento:"code-dump-exporter@1.0.0"},
  {id:"pre",      elemento:"pre-procesar→estructura@1.x"},
  {id:"struct",   elemento:"constructor-de-estructuras@1.x"},
  {id:"analysis", elemento:"analisis-estructural@1.x"},
  {id:"qa",       elemento:"validador-invariantes@1.x"},
  {id:"publish",  elemento:"publicador-cn@1.x"}
]

# SALIDAS
salidas: [
  "CognitiveNodePackage (TXT/ZIP) con {contexto, nodes, vistas, tesis, plantillas,
evidencias, provenance}",
  "project-dump.jsonl + MANIFEST.json (si aplica)",
  "qa.report, metrics.log, provenance.log"
]
```

```
# MENSAJERÍA (prompts-guía por puerta)
mensajes: {
  P1: "Por favor envía:  Contrato = { ...schema JSONL... }  (puedes pegar el tuyo o confirmar
el base).",
  P2: "Indica el material de entrada:  Entrada = { tipo:'repo'|'contrato', detalle:… }  y
Exclusiones = ['…'].",
  P3: "Configura invariantes y confianza:  Invariantes = {...}  Confianza = 0.7",
  P4: "Propuesta de plan (modo y pasos). Responde: OK plan  o  Ajustar(…)",
  P5: "Exporter/scaffold. Responde:  Exporter = { --out:'project-dump.jsonl', … }  o  Scaffold =
{ lenguaje:'…' }",
  P6: "Parámetros de pre-proceso:  Pre = { maxInvalidRate:0.5 }",
  P7: "Constructor de estructuras:  Struct = { segmentacion:'archivo|módulo', … }",
  P8: "Análisis estructural:  Analisis = { ejes:[…], macro:'…' }",
  P9: "QA:  QA = { minScore:0.8 }",
  P10:"Publicación:  Publicar = { destino:'download'|ruta, formato:'txt'|'zip' }"
}

# ERRORES ESTÁNDAR Y ACCIONES
errores: [
  {code:"E_CONTRATO", tip:"Contrato ausente o inválido", accion:"reanclar a P1 y mostrar
ejemplo válido"},
  {code:"E_MATERIAL", tip:"Entrada no especificada",     accion:"reanclar a P2"},
  {code:"E_PLAN",     tip:"Plan no confirmado",          accion:"reanclar a P4"},
  {code:"E_QA",       tip:"Confianza < umbral",          accion:"sugerir ajustes en P6–P8 o
elevar umbral con justificación"}
]

# SEGURIDAD / PRIVACIDAD
seguridad: {
  excluir_por_defecto: [".git/**","**/node_modules/**","**/secrets/**",".env","**/dist/**"],
  no_leer_credenciales: true,
  confirmar_fuentes_sensibles: true
}

# CRITERIO DE FINALIZACIÓN
exito_si: [
  "CognitiveNodePackage emitido",
  "QA ≥ umbral_confianza",
  "Provenance completo"
]


# ------------------------------------------------------------------------
# END OF CONSCIENCE BLOCK: COGNITIVE MODE: CREATION (desde dump JSONL)
# ------------------------------------------------------------------------
```