

MindScriptDoc

MindScript Language Reference v0.1

MindScript is a human-readable scripting language for describing mental processes, behavioral patterns, emotional responses, decision rules, social dynamics, and learned psychological scripts.

Its design goal is simple:

Readable by non-programmers, familiar to programmers, structured enough for software to interpret.

MindScript combines:

- Java-like function syntax
- SQL-like uppercase logic keywords
- ordinary English
- simple algebra
- indentation for readability
- braces for script boundaries

The language intentionally avoids advanced programming syntax where a simpler human-readable alternative exists.

1. Basic Script Structure

Every behavioral process is represented as a script.

```
scriptName(parameters) {
```

```
    PURPOSE:  
    ...
```

```
    REQUIRES:  
    ...
```

```
    NEEDS:  
    ...
```

```
    BELIEF:  
    ...
```

```
    WHEN:  
    ...
```

```
    WHERE:
```

```

        ...
DO:
    ...
IF condition:
    ...
ELSE:
    ...
WHILE condition:
    ...
RETURN:
    ...
RISKS:
    ...
REPLACEMENT:
    ...
}

```

Not every section is required.

A minimal script may be:

```

comfortResponse(person) {
    WHEN:
        personIsDistressed

    DO:
        offerSupport()

    RETURN:
        reassurance
}

```

2. Naming

Scripts

Scripts always use parentheses.

```

evaluateThreat()
buildTrust()
offerAdvice(person)
accountabilitySelector(problem)

```

Parentheses mean:

This is something that runs.

CamelCase is recommended.

```
missionImpossibleDrive()  
correctionAsCaring()  
offerConfirmReturnAgency()
```

Variables

Variables do not use parentheses.

```
fear  
trust  
pain  
dangerLevel  
selfConfidence  
adviceStatus
```

Variables represent state or stored information.

Example:

```
fear = high  
trust = low  
dangerLevel = immediate
```

Values

Values are simple descriptive words.

```
low  
medium  
high  
veryHigh
```

```
yes  
no
```

```
requested  
declined  
unknown
```

```
immediate  
moderate  
remote
```

Example:

```
confidence = high  
dangerLevel = immediate  
adviceStatus = declined
```

Quotation marks are used when the value is natural-language text:

```
BELIEF:
    "If I care about someone, I should prevent them from making mistakes."
```

3. Comments

MindScript uses Java-style comments.

Single-line comment:

```
// This is a comment
```

Multi-line comment:

```
/*
    This is a longer comment.

    Comments do not affect script execution.
*/
```

Comments are useful for:

- explanations
- uncertainty
- historical context
- examples
- author notes
- alternative interpretations

Example:

```
// Hypothesis rather than established fact
BELIEF:
    "If they understood my reasoning, they would agree."
```

4. PURPOSE

PURPOSE describes what the script is trying to accomplish.

```
correctionAsCaring(person) {
    PURPOSE:
        protect person from avoidable mistakes
}
```

The purpose may differ from the actual outcome.

For example:

```
PURPOSE:
  protect person from harm
```

```
RISKS:
  personFeelsControlled
  relationshipConflict
```

This distinction is important because a script can have benevolent intent while producing undesirable results.

5. REQUIRES

REQUIRES lists other scripts that this script depends upon.

```
missionImpossibleDrive() {
  REQUIRES:
    grandVision()
    valueComparison()
    evaluateReality()
}
```

A dependency is another process.

The rule is:

```
something() = script/process
something   = state/data
```

6. NEEDS

NEEDS describes prerequisites or internal conditions required for the script to operate.

```
missionImpossibleDrive() {
  NEEDS:
    selfConfidence > high
    beliefVisionIsPossible = veryHigh
}
```

Difference:

```
REQUIRES = another script is needed
NEEDS    = some state must already exist
```

Example:

```
correctionAsCaring(person) {  
    REQUIRES:  
        evaluateRisk()  
  
    NEEDS:  
        concernForPerson > medium  
}
```

7. WHEN

WHEN defines the event or condition that activates a script.

```
angerResponse(event) {  
    WHEN:  
        perceivedViolation  
}
```

Multiple triggers may be combined:

```
WHEN:  
    perceivedMistake  
    AND concernForPerson > medium
```

WHEN answers:

What causes this script to start?

8. WHERE

WHERE narrows the context in which the script applies.

```
offerAdvice(person) {  
    WHEN:  
        potentialProblemDetected  
  
    WHERE:  
        relationship = close  
        OR consequences > significant  
}
```

WHEN defines activation.

WHERE defines applicability.

Example:

```
comfortResponse(person) {  
    WHEN:  
        personIsDistressed  
  
    WHERE:  
        relationship = friend  
        OR relationship = family  
}
```

9. DO

DO contains actions performed by the script.

```
bridge(problem) {  
    DO:  
        acknowledgeMyRole()  
        validateImpact()  
        ask("What needs to happen next?")  
        fixWhatICan()  
}
```

Actions normally execute from top to bottom.

State changes may also occur inside DO.

```
DO:  
    trust = trust + medium  
    fear = fear - low
```

10. IF

IF performs conditional logic.

```
IF dangerLevel = immediate:  
    intervene()
```

Multiple conditions may be combined:

```
IF personUnderstands AND adviceStatus = declined:  
    returnAgency()
```

MindScript uses ordinary English logical operators.

```
AND  
OR  
NOT
```

Example:

```
IF anger > high AND NOT immediateDanger:  
    pauseBeforeResponding()
```

11. ELSE

ELSE specifies the alternative behavior when an IF condition is not met.

```
IF adviceStatus = requested:  
    explainPerspective()
```

```
ELSE:  
    returnAgency()
```

ELSE IF may be used:

```
IF dangerLevel = immediate:  
    intervene()
```

```
ELSE IF dangerLevel = high:  
    warnStrongly()
```

```
ELSE:  
    offerPerspective()
```

12. WHILE

WHILE repeats behavior while a condition remains true.

```
WHILE validation < enough:
```

```
    DO:  
        seekReassurance()
```

Example:

```
WHILE personDisagrees AND hopeOfInfluence > low:
```

```
    DO:  
        strongerArgument()
```

Loops are useful for representing self-reinforcing behavioral cycles.

13. RETURN

RETURN represents the result produced by a script.

```
evaluateThreat(event) {  
    RETURN:  
        dangerLevel  
}
```

A script can also return another script's result:

```
accountabilitySelector(problem) {  
    IF punishmentForMistakes > high:  
        RETURN:  
            shield(problem)  
  
    ELSE:  
        RETURN:  
            bridge(problem)  
}
```

RETURN differs from state changes.

Example:

```
comfortBeacon() {  
    DO:  
        selfEsteem = selfEsteem + medium  
  
    RETURN:  
        relief  
}
```

selfEsteem changed as a side effect.

relief is what the script returned.

14. State Changes

MindScript uses ordinary algebra rather than programmer-specific operators.

Preferred:

```
fear = fear + medium  
trust = trust - low
```

Avoid:

```
fear += medium  
trust -= low
```

The goal is for someone without programming experience to understand the expression.

15. Comparisons

MindScript uses simple mathematical comparisons.

```
=  
>  
<  
>=  
<=
```

Examples:

```
IF fear > high:  
    withdraw()
```

```
IF confidence = low:  
    seekReassurance()
```

```
IF trust <= medium:  
    avoidDisclosure()
```

MindScript does not require ==.

Context determines whether = means assignment or comparison.

Example:

```
fear = high
```

sets a value.

```
IF fear = high:
```

tests a value.

16. Logical Operators

MindScript uses SQL-style words instead of programming symbols.

```
AND  
OR  
NOT
```

Preferred:

```
IF trust > medium AND fear < high:
```

instead of:

```
if (trust > medium && fear < high)
```

Preferred:

```
IF NOT adviceWanted:
```

instead of:

```
if (!adviceWanted)
```

17. BELIEF

BELIEF describes a belief that drives the script.

```
shield(problem) {  
    BELIEF:  
        "Accepting responsibility means accepting punishment."  
}
```

A script may contain multiple beliefs:

```
BELIEFS:  
    "Mistakes reduce my social value."  
    "Being blamed means losing control."
```

Beliefs may be explicit or unconscious.

18. LEARNED

LEARNED describes possible origins of a script.

```
correctionAsCaring() {  
    LEARNED:  
        repeatedExposureToCorrectionAsCare  
        watchingAuthorityFiguresCorrectOthers  
        beingProvenRightAfterWarnings  
}
```

This does not imply certainty.

When origins are speculative, comments should clarify this:

```
// Possible developmental origin  
LEARNED:  
    intermittentAttention
```

19. ASSUMES

ASSUMES describes interpretations that are not directly observed.

```
prestigeBrandElevation(person) {  
    ASSUMES:  
        attentionScarcityProvidesValidation  
        distanceMayIncreasePursuit  
}
```

This is useful when modeling other people.

It prevents interpretations from being confused with facts.

Optional confidence can be added:

```
ASSUMES:  
    validationSeeking  
        confidence = medium  
  
    deliberateManipulation  
        confidence = low
```

20. OBSERVED

OBSERVED contains directly observable behavior.

```
OBSERVED:  
    attentionIsIntermittent  
    communicationDecreasesAfterIntimacy
```

This allows a useful separation:

```
OBSERVED = what happened  
ASSUMES  = what we think it means
```

21. RISKS

RISKS describes unintended or maladaptive consequences.

```
correctionAsCaring() {  
    RISKS:  
        adviceBecomesPressure
```

```
        disagreementBecomesContest
        personFeelsControlled
    }
```

A script may be useful and still contain risks.

22. REPLACEMENT

REPLACEMENT points to a healthier or more appropriate script.

```
correctionAsCaring(person) {
    REPLACEMENT:
        offerConfirmReturnAgency(person)
}
```

This makes MindScript useful not merely for analysis, but for behavioral modification.

23. State vs. Process

One of the fundamental MindScript distinctions is:

```
state    = what currently exists
script   = what changes or evaluates state
```

Examples:

```
fear
anger
love
pain
trust
```

are states.

Processes have parentheses:

```
generateFear()
generateAnger()
buildTrust()
processPain()
evaluateThreat()
```

Example:

```
anger = generateAnger(event)
```

24. Emotions

Emotions may exist as state variables.

```
STATE:
    anger = low
    fear = medium
    sadness = low
    attachment = high
```

But the mechanism that creates the emotion should normally be modeled as a script.

Example:

```
generateAnger(event) {

    REQUIRES:
        evaluateViolation()
        assignResponsibility()

    DO:
        violation = evaluateViolation(event)
        responsibility = assignResponsibility(event)

    RETURN:
        violation * responsibility * importance
}
```

Then:

```
anger = generateAnger(event)
```

25. Composite Emotions

Some emotional concepts may represent several underlying systems.

For example:

```
romanticLove(person) {

    REQUIRES:
        attachment(person)
        attraction(person)
        buildTrust(person)
        intimacy(person)

    RETURN:
        attachmentLevel
        + attractionLevel
        + intimacyLevel
        + admirationLevel
        + careLevel
}
```

This means `love` can exist as a state while `romanticLove()` describes the process producing it.

26. VALUES

Scripts may optionally define allowed values.

VALUES:

```
dangerLevel =  
    low,  
    moderate,  
    high,  
    immediate
```

```
confidence =  
    low,  
    medium,  
    high
```

```
adviceStatus =  
    unknown,  
    requested,  
    declined
```

This section is optional.

MindScript should remain usable without formal variable declarations.

27. STATE

STATE can describe the current internal condition of a person, system, relationship, or organization.

STATE:

```
trust = medium  
anger = low  
selfConfidence = high  
fear = low
```

State may change as scripts run.

28. Example: Correction as Caring

```
correctionAsCaring(person) {
```

```
    PURPOSE:  
        protect person from avoidable mistakes
```

```

REQUIRES:
    evaluateRisk()
    valueComparison()

NEEDS:
    concernForPerson > medium

LEARNED:
    correctionUsedAsCare
    repeatedExperienceOfBeingProvenRight

BELIEF:
    "If I can see a danger, I should help them avoid it."

WHEN:
    perceivedMistake

DO:
    explainRisk()
    offerAlternative()

IF personDisagrees:

    DO:
        frustration = frustration + medium

    IF hopeOfInfluence > low:
        explainAgain()

WHILE personDisagrees AND hopeOfInfluence > low:

    DO:
        strongerArgument()

RETURN:
    warningDelivered

RISKS:
    adviceBecomesPressure
    disagreementBecomesContest
    personFeelsControlled
    goalChangesFromUnderstandingToAgreement

REPLACEMENT:
    offerConfirmReturnAgency(person)
}

```

29. Example: Offer, Confirm, Return Agency

```

offerConfirmReturnAgency(person) {

    PURPOSE:
        help without taking ownership of another adult's decisions

    WHEN:

```

```

    potentialHarmDetected
DO:
    dangerLevel = evaluateDanger()
IF dangerLevel = immediate:
    DO:
        intervene()
    RETURN:
        immediateProtection
ELSE:
    DO:
        adviceStatus = askIfAdviceWanted()
    IF adviceStatus = requested:
        DO:
            explainPerspective()
            understanding = confirmUnderstanding()
        IF understanding = confirmed:
            DO:
                returnAgency()
    ELSE:
        DO:
            returnAgency()
RETURN:
    autonomyPreserved
}

```

30. Example: Accountability Selector

```

accountabilitySelector(problem) {
    REQUIRES:
        shield(problem)
        bridge(problem)
    WHEN:
        problemDetected
    IF punishmentForMistakes > high:
        RETURN:
            shield(problem)
    ELSE IF psychologicalSafety > high:

```

```
        RETURN:
            bridge(problem)

    ELSE:

        RETURN:
            evaluateContext(problem)
}
```

31. Example: Shield

```
shield(problem) {

    PURPOSE:
        protect self from blame

    BELIEF:
        "Accepting responsibility means accepting punishment."

    WHEN:
        mistakeDetected

    DO:
        minimizeMyRole()
        explainBeforeSolving()
        shiftBlame()

    RETURN:
        temporarySelfProtection

    RISKS:
        trust = trust - high
        resolutionSpeed = resolutionSpeed - medium
        repeatFailureRisk = repeatFailureRisk + high
}
```

32. Example: Bridge

```
bridge(problem) {

    PURPOSE:
        resolve problem and restore trust

    BELIEF:
        "Accepting responsibility gives me power to fix things."

    WHEN:
        mistakeDetected

    DO:
```

```
        acknowledgeMyRole()
        validateImpact()
        ask("What needs to happen next?")
        fixWhatICan()

    RETURN:
        problemResolution

    RISKS:
        temporaryDiscomfort
}
```

33. Example: Comfort Beacon

```
comfortBeacon() {
    PURPOSE:
        restore self-esteem through external reassurance

    WHEN:
        selfEsteem < comfortable

    WHILE validation < enough:

        DO:
            selfDenigration()
            seekComfort()

    RETURN:
        temporaryRelief

    RISKS:
        dependenceOnExternalValidation
        escalatingSignals
}
```

34. Example: Dictatorship Dilemma

```
dictatorshipDilemma(organization) {
    NEEDS:
        concentratedAuthority > high
        honestFeedback < low

    WHEN:
        leaderReceivesBadNews
        OR leaderReceivesCriticism

    DO:
        hostileResponse()
}
```

```

    teamFear = teamFear + high
    informationHonesty = informationHonesty - high

WHILE teamFear > psychologicalSafety:
    DO:
        filterBadNews()

        informationQuality =
            informationQuality - medium

        decision =
            makeDecision(informationQuality)

    IF reality != expectation:
        DO:
            blameOthers()

            teamFear = teamFear + high
            informationHonesty = informationHonesty - high

RETURN:
    brittleOrganization
}

```

Note: future versions should replace != here with:

```
IF NOT reality = expectation:
```

or preferably:

```
IF reality NOT = expectation:
```

However, the final preferred syntax for inequality remains to be standardized.

35. Recommended Inequality Syntax

Because != may not be familiar to non-programmers, MindScript should preferably use ordinary language.

Preferred:

```
IF reality is NOT expectation:
```

or:

```
IF reality NOT = expectation:
```

The first is easier to read.

Therefore MindScript may eventually allow relational English:

IF dangerLevel is high:
IF person is NOT ready:
IF trust is below medium:
IF confidence is above high:

This remains under consideration for future versions.

36. Core Design Rule

MindScript follows one overriding rule:

If understanding a symbol requires programming experience, prefer ordinary language.

Therefore:

AND
OR
NOT

are preferred over:

&&
||
!

And:

fear = fear + medium

is preferred over:

fear += medium

37. Conceptual Architecture

MindScript treats a person or system as interacting layers.

memories
 stored experiences

beliefs
 persistent interpretations

values
 weighting and priorities

state
 current emotional and cognitive condition

scripts

learned processes

triggers

events that start scripts

actions

behaviors produced by scripts

feedback

consequences that strengthen or weaken scripts

A simplified mental architecture could be written as:

experience

->

belief

->

script

->

behavior

->

outcome

->

learning

->

updated belief/script

38. Fundamental Philosophy

MindScript describes behavior as something a person runs rather than something a person inherently is.

Instead of:

“I am controlling.”

MindScript encourages:

correctionAsCaring()

Meaning:

“Under certain conditions, I run a script that can become controlling.”

Instead of:

“She is avoidant.”

MindScript encourages:

distanceResponse()

Meaning:

“Under certain conditions, a distancing process appears to activate.”

This distinction allows scripts to be:

- inspected
- questioned
- modified
- replaced
- combined
- weakened
- reinforced

Personality therefore becomes less like a fixed label and more like a collection of interacting processes.

39. MindScriptDoc Comments

Individual scripts may optionally begin with documentation comments similar to JavaDoc.

```
/**
 * Protects another person from a perceived avoidable mistake.
 *
 * Intended purpose:
 *     reduce preventable harm
 *
 * Common failure mode:
 *     advice becomes pressure when disagreement continues
 *
 * Suggested replacement:
 *     offerConfirmReturnAgency()
 */
correctionAsCaring(person) {
    ...
}
```

These documentation comments are called **MindScriptDoc comments**.

They use:

```
/**
 *
 */
```

Regular implementation comments continue to use:

```
// comment

/*
```

```
comment
*/
```

40. MindScriptDoc Tags

Optional JavaDoc-inspired tags may eventually be supported.

Example:

```
/**
 * Attempts to prevent another person from making a harmful decision.
 *
 * @purpose prevent avoidable harm
 * @trigger perceived mistake
 * @requires evaluateRisk()
 * @risk person may feel controlled
 * @replacement offerConfirmReturnAgency()
 */
correctionAsCaring(person) {
    ...
}
```

Possible MindScriptDoc tags:

```
@purpose
@param
@requires
@trigger
@belief
@returns
@risk
@replacement
@learned
@example
@see
```

Example:

```
/**
 * @purpose Preserve autonomy while making useful information available.
 * @param person Person receiving the advice.
 * @returns autonomyPreserved
 * @see correctionAsCaring()
 */
offerConfirmReturnAgency(person) {
    ...
}
```

41. MindScript v0.1 Reserved Keywords

Current reserved keywords:

PURPOSE
REQUIRES
NEEDS
VALUES
STATE

OBSERVED
ASSUMES
BELIEF
BELIEFS
LEARNED

WHEN
WHERE

IF
ELSE
WHILE
DO

AND
OR
NOT

RETURN

RISKS
REPLACEMENT

Potential future keywords:

EVENTUALLY
UNTIL
FOR EACH
ONCE
AFTER
BEFORE
WEIGHT
CONFIDENCE
REINFORCED BY
WEAKENED BY

These should not be added unless they solve a recurring modeling problem.

42. Design Principles

MindScript should remain:

Human-readable

A non-programmer should understand most scripts without documentation.

Executable-looking

A programmer should immediately understand the causal structure.

Psychologically humble

Observed facts should remain distinguishable from inferred motives.

Composable

Small scripts should combine into larger scripts.

Modifiable

Maladaptive scripts should be able to point toward replacement scripts.

Non-diagnostic

MindScript describes processes and hypotheses rather than assigning fixed labels to people.

Simple

Language complexity should only be added when simpler constructs cannot express the idea.

43. The Core MindScript Pattern

Most scripts can ultimately be represented as:

```
scriptName(input) {  
    REQUIRES:  
        otherScripts()  
  
    NEEDS:  
        existingState  
  
    BELIEF:  
        "Underlying assumption"  
  
    WHEN:  
        trigger  
  
    WHERE:  
        context  
  
    DO:  
        actions()  
}
```

```

IF condition:
    alternativeAction()

WHILE condition:
    repeatedAction()

RETURN:
    result

RISKS:
    unwantedConsequences

REPLACEMENT:
    healthierScript()
}

```

This is the conceptual heart of MindScript.

44. One-Line Mental Model

MindScript can be summarized as:

**Memories install beliefs. Beliefs activate scripts. Scripts modify state and behavior.
Outcomes reinforce or weaken the scripts.**

Or, in MindScript form:

```

humanLearning(experience) {
    DO:
        belief = interpret(experience)
        script = learn(belief)
        behavior = script()

    AFTER:
        outcome = observeResult(behavior)
        update(script, outcome)

    RETURN:
        updatedBehaviorModel
}

```

Future versions may formally introduce AFTER.