DAKSH

User Guide & Language Reference

Phase 0 — Web Interpreter Edition

Purpose: A practical guide to writing, understanding, and running Daksh programs.

Document status: Phase 0 baseline / current web-interpreter target

1. What is Daksh?

Daksh is a programming language designed around a simple, readable syntax. The current implementation targets a browser-based interpreter. A Daksh source file uses the **.dax** extension.

Current execution pipeline:

```
Daksh Source
    ↓
Lexer
    ↓
Tokens
    ↓
Parser
    ↓
AST
    ↓
Interpreter
    ↓
Runtime
    ↓
Output
```

This guide documents the Phase 0 language foundation that has been implemented and tested in the current web interpreter.

2. Your First Daksh Program

```
tell "Hello, Daksh!"
```

The **tell** statement prints a value to the output panel.

```
set name = "Manideep"
tell name
```

A Daksh program can contain statements one after another without requiring semicolons.

3. Variables

Use **set** to declare a variable or change its value. Daksh variables are dynamically typed.

```
set name = "Manideep"
set age = 21
set score = 95.5

set age = 22
```

Multiple declarations are supported:

```
set name = "Manideep", age = 21, score = 95.5
```

Compound assignment is supported:

```
set x = 10
set x += 5
set x -= 3
set x *= 2
set x /= 4
set x %= 5
```

Constants use const and cannot be reassigned:

```
const pi = 3.14
tell pi

set pi = 3.14159   // runtime error
```

Variable names are case-sensitive and follow the language's identifier rules. Keywords are reserved.

4. Data Types

The Phase 0 data model includes Integer, Float/Decimal, String/Text, Boolean, Null, List/Array, and Object/Dictionary. Character and Tuple remain optional.

```
set age = 21
set price = 95.5
set name = "Daksh"
set active = true
set result = null
set numbers = [10, 20, 30]
set user = {
    name: "Manideep",
    age: 21
}
```

Daksh supports dynamic typing:

```
set value = 10
set value = "Daksh"
set value = true
set value = [1, 2, 3]
```

Type conversion and inspection:

```
set age = integer("25")
set textValue = text(100)

tell type(age)
tell type(textValue)
```

5. Lists and Objects

Lists use square brackets and are indexed from zero. Objects use key/value pairs.

```
set numbers = [10, 20, 30]
tell numbers[0]
tell numbers[2]

set user = {
    name: "Daksh",
    age: 21
}
tell user.name
```

Nested structures are supported:

```
set data = {
    user: {
        name: "Manideep",
        scores: [85, 90, 95]
    }
}

tell data.user.name
tell data.user.scores[2]
```

Nested assignment is supported:

```
set data.user.name = "Daksh"
set data.user.scores[1] = 99

set matrix = [
    [1, 2],
    [3, 4]
]

set matrix[0][1] = 99
```

6. Arithmetic

Arithmetic operators:

+ addition, **-** subtraction, ***** multiplication, **/** division, **%** remainder, ****** exponent.

```
set a = 10
set b = 3

tell a + b
tell a - b
tell a * b
tell a / b
tell a % b
tell a ** b
```

Parentheses and operator precedence are supported:

```
set a = 2 + 3 * 4
set b = (2 + 3) * 4

tell a
tell b
```

Unary minus is also supported.

7. Conditions

Use **if**, **else if**, and **else** for conditional execution.

```
set score = 85

if score >= 90 {
    tell "A"
} else if score >= 75 {
    tell "B"
} else {
    tell "C"
}
```

Comparison operators: ==, !=, <, >, <=, >=. Logical operators: &&, ||, !.

```
set age = 21
set active = true

if age >= 18 && active {
    tell "Allowed"
}
```

Ternary and pattern matching are future features, not part of the current Phase 0 implementation.

8. Loops

Daksh supports **while**, **loop**, **for**, and **foreach**.

```
set i = 1

while i <= 5 {
    tell i
    i++
}

for i in 1..5 {
    tell i
}

set numbers = [10, 20, 30]

foreach number in numbers {
    tell number
}
```

Loop control:

```
set i = 1

while i <= 10 {
    if i == 5 {
        break
    }

    if i == 3 {
        i++
        continue
    }

    tell i
    i++
}
```

Increment and decrement operators are **++** and **--**.

9. Functions

Declare functions with **function**. Parameters and return values are supported.

```
function add(a, b) {
    return a + b
}

tell add(10, 20)
```

Functions support local variables and recursion:

```
function factorial(n) {
    if n <= 1 {
        return 1
    }

    return n * factorial(n - 1)
}

tell factorial(5)
```

Functions can return lists and objects and can be combined with loops and standard-library calls.

Default parameters, named parameters, multiple return values, and anonymous/lambda functions are not finalized in the current Phase 0 syntax.

10. Operators

Assignment: **=**. Arithmetic: **+ - * / % ****. Comparison: **== != < > <= >=**. Logical: **&& || !**. Compound assignment: **+= -= *= /= %=**. Increment/decrement: **++ --**.

Identity and membership operators were considered in the design but their exact Daksh syntax is not finalized. Bitwise and ternary operators are future work.

11. Keywords

Core Phase 0 keywords:

```
tell
set
const
input
if
else
loop
while
for
break
continue
function
return
import
true
false
null
foreach
```

Keywords are reserved and Daksh is case-sensitive.

12. Expressions

Expressions can contain literals, variables, arithmetic, comparisons, logical operations, function calls, indexing, property access, and nested combinations.

```
function square(n) {
    return n ** 2
}

set result = square(5) + 10
tell result

set data = {
    user: {
        scores: [10, 20, 30]
    }
}

tell data.user.scores[1]
```

The current language treats assignment as a statement rather than a general expression.

13. Statements and Blocks

Daksh statements include declarations, assignments, output, input, conditions, loops, function declarations, returns, breaks, continues, and imports. Curly braces define blocks for functions, conditions, and loops.

```
set name = "Daksh"

if true {
    tell name

    foreach item in [1, 2, 3] {
        tell item
    }
}
```

14. Input and Output

tell writes to the interpreter output. **input()** requests user input in the browser runtime.

```
set name = input("Enter your name: ")
tell name

set age = integer(
    input("Enter your age: ")
)

tell age
```

15. Modules and Standard Library

Use **import** to load standard modules. Current modules are math, string, list, random, file, time, and system.

```
import math
import string
import list

tell math.sqrt(144)
tell string.upper("daksh")

set numbers = [10, 20, 30]
tell list.length(numbers)
```

Math module:

```
abs  sqrt  pow  floor  ceil  round  max  min  pi  e
```

String module:

```
length  upper  lower  trim  contains  startsWith  endsWith  replace
```

List module:

```
length  add  remove  contains  first  last
```

Random module:

```
number  integer  choice
```

File module:

```
read  write  download  exists
```

Time module:

```
today()
```

```
now()
nowDateTime()
format(date, pattern)
addDays(date, days)
removeDays(date, days)
isBefore(date1, date2)
isAfter(date1, date2)
dayName(date)
monthName(date)
yearNumber(date)
monthNumber(date)
dayNumber(date)
```

System module:

```
osName()
osVersion()
deviceName()
userName()
language()
architecture()
dakshVersion()
memory()
batteryLevel()
isOnline()
sleep(seconds)
exit()
restart()
```

File access is browser-safe: arbitrary local-path inspection is restricted by browser security. The current **exists()** behavior reflects that limitation.

16. Time Module — Examples

```
import time

tell time.today()
tell time.now()
tell time.nowDateTime()

set today = time.today()
tell time.format(today, "YYYY-MM-DD")
tell time.dayName(today)
tell time.monthName(today)
```

The exact formatting tokens supported in the current implementation are YYYY, MM, DD, HH, mm, and ss.

17. System Module — Examples

```
import system

tell system.osName()
tell system.osVersion()
tell system.deviceName()
tell system.userName()
tell system.language()
tell system.architecture()
tell system.dakshVersion()
tell system.memory()
tell system.batteryLevel()
tell system.isOnline()
```

sleep(seconds) is asynchronous in the browser runtime. exit() and restart() are runtime-level operations and may be constrained by browser security and environment behavior.

18. Error Handling

The interpreter reports syntax and runtime problems instead of silently continuing. Current error categories include syntax errors, runtime errors, undefined variables, division by zero, invalid indexes, import errors, and constant modification.

```
set numbers = [10, 20, 30]
set numbers[10] = 100
```

The above should produce a Daksh runtime error for an out-of-range list index.

```
const x = 10
set x = 20
```

This should produce a constant-modification runtime error.

Custom errors are planned but their final syntax is not part of the Phase 0 baseline.

19. Scope and Memory Model

Variables in an outer scope are visible inside nested blocks. Variables created in an inner block are not visible outside that block. Function calls have their own local environment.

```
set name = "Manideep"

if true {
    set message = "Hello"
    tell name
    tell message
}

tell name
```

The current web interpreter relies on the JavaScript runtime for actual memory management. A native Daksh heap/stack/garbage-collection design remains future work.

20. Complete Example

The following combines modules, functions, strings, lists, loops, conditions, objects, and assignment.

```
import math
import string
import list

function processStudent(name, marks) {
    set cleanName = string.upper(string.trim(name))
    set total = 0

    foreach mark in marks {
        set total += mark
    }

    set average = total / list.length(marks)

    if average >= 90 {
        return {
            name: cleanName,
            average: average,
            grade: "A"
```

```
        }
    } else if average >= 75 {
        return {
            name: cleanName,
            average: average,
            grade: "B"
        }
    } else {
        return {
            name: cleanName,
            average: average,
            grade: "C"
        }
    }
}

set marks = [85, 90, 95]

set student = processStudent(
    "  manideep  ",
    marks
)

tell student.name
tell student.average
tell student.grade

set student.grade = "A+"
tell student.grade

tell math.sqrt(144)
```

Expected output for this example:

```
MANIDEEP
90
A
A+
12
```

21. Current Phase 0 Status

Phase 0 is the frozen language-foundation baseline for the current web-interpreter target.

```
Variables            ■
Data Types           ■
Arithmetic           ■
Conditions           ■
Loops                ■
Functions            ■
Operators            ■
Keywords             ■
Expressions          ■
Statements           ■
Modules              ■
Error Handling       ■
Input / Output       ■
Standard Library     ■
Web Interpreter      ■
```

Memory architecture and native compiler/runtime work remain later-stage concerns. Some syntax families—such as custom errors, module exports/aliases, default/named function parameters, multiple returns, identity/membership operators, and certain future operators—were intentionally left unfinished rather than inventing syntax.

22. Quick Reference

```
# Variables
set x = 10
const y = 20

# Multiple variables
set name = "Daksh", age = 21

# Output / input
tell "Hello"
set name = input("Name: ")

# Conditions
if x > 5 {
    tell "yes"
} else {
    tell "no"
}

# Loops
while x < 10 {
    x++
}

for i in 1..5 {
    tell i
}

foreach item in [1, 2, 3] {
    tell item
}

# Functions
function add(a, b) {
    return a + b
}

tell add(2, 3)

# Objects
set user = {
    name: "Daksh",
    age: 21
}

set user.age = 22

# Lists
set numbers = [10, 20, 30]
set numbers[1] = 99

# Modules
import math
tell math.sqrt(144)
```

23. Recommended Way to Learn Daksh

1. Start with variables and tell. 2. Learn data types and lists/objects. 3. Learn arithmetic and conditions. 4. Practice loops. 5. Learn functions and recursion. 6. Learn modules and standard-library APIs. 7. Build small programs combining several features. 8. Use the error messages to debug your code.

Suggested first projects: calculator, number guessing game, student grade calculator, todo/list manager, simple text processor, and a small data-processing program.

24. Phase 0 Notes

This document is a user guide for the currently implemented Phase 0 web interpreter. It intentionally does not invent syntax for features whose exact design was not finalized. Native compiler, CLI, VS Code integration, and a native runtime are later-stage work.

Daksh — User Guide & Language Reference