DAKSH

Example Programs Collection

Phase 0 — Practice & Demonstration Programs

A separate practical companion to the Daksh User Guide. This PDF focuses on programs, not language-reference explanations.

Goal: Practice Daksh by running examples from basic syntax through combined Phase 0 programs.

How to Use This Book

Run the programs in order. The early examples isolate one concept; later examples intentionally combine multiple concepts. Compare the output with the expected output and modify the code to experiment.

The examples use only the currently established Phase 0 syntax and the currently implemented standard-library APIs. Features whose exact syntax was not finalized are not invented here.

PART I — BASICS

1. Hello World

What it demonstrates: Basic output with tell.

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

Expected output:

```
Hello, Daksh!
```

2. Variables

What it demonstrates: Variable declaration and reading values.

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

tell name
tell age
```

Expected output:

```
Manideep
21
```

3. Multiple Variables

What it demonstrates: Multiple declarations in one statement.

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

tell name
tell age
tell score
```

Expected output:

```
Manideep
21
95.5
```

4. Dynamic Typing

What it demonstrates: Changing a variable's value to different types.

```
set value = 10
tell value

set value = "Daksh"
tell value

set value = true
tell value

set value = [1, 2, 3]
tell value
```

5. Constants

What it demonstrates: Declaring and protecting a constant.

```
const pi = 3.14

tell pi

# This produces a runtime error:
# set pi = 3.14159
```

6. Type Conversion

What it demonstrates: Converting input/text values.

```
set age = integer("21")
set score = integer("95")
set message = text(100)

tell age
tell score
tell message
```

Expected output:

```
21
95
100
```

7. Type Inspection

What it demonstrates: Using type().

```
set a = 10
set b = "Daksh"
set c = true
set d = [1, 2, 3]

tell type(a)
tell type(b)
tell type(c)
tell type(d)
```

PART II — ARITHMETIC & OPERATORS

8. Arithmetic Calculator

What it demonstrates: Basic arithmetic operators.

```
set a = 20
set b = 6

tell a + b
tell a - b
tell a * b
tell a / b
tell a % b
tell a ** 2
```

Expected output:

```
26
14
120
3.3333333333333335
2
400
```

9. Precedence

What it demonstrates: Understanding precedence and parentheses.

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

tell a
tell b
```

Expected output:

```
14
20
```

10. Compound Assignment

What it demonstrates: Using +=, -=, *=, /= and %=.

```
set x = 10

set x += 5
tell x

set x -= 3
tell x

set x *= 2
tell x

set x /= 4
tell x

set x %= 5
tell x
```

Expected output:

```
15
12
24
6
1
```

11. Increment and Decrement

What it demonstrates: Using ++ and -- in a loop.

```
set i = 1

tell i
i++
tell i
i++
tell i
i--
tell i
```

Expected output:

```
1
2
3
2
```

PART III — CONDITIONS

12. Simple If

What it demonstrates: Basic condition.

```
set age = 21

if age >= 18 {
```

```
    tell "Adult"
}
```

Expected output:

```
Adult
```

13. If / Else

What it demonstrates: Two-way decision.

```
set number = 7

if number % 2 == 0 {
    tell "Even"
} else {
    tell "Odd"
}
```

Expected output:

```
Odd
```

14. Else If Grading

What it demonstrates: Multiple conditions.

```
set score = 85

if score >= 90 {
    tell "A"
} else if score >= 75 {
    tell "B"
} else if score >= 60 {
    tell "C"
} else {
    tell "Fail"
}
```

Expected output:

```
B
```

15. Logical Operators

What it demonstrates: Combining conditions.

```
set age = 21
set active = true

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

if age < 18 || !active {
    tell "Restricted"
}
```

Expected output:

```
Allowed
```

PART IV — LOOPS

16. While Loop

What it demonstrates: Repeating while a condition is true.

```
set i = 1
```

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

Expected output:

```
1
2
3
4
5
```

17. For Range

What it demonstrates: Inclusive range loop.

```
for i in 1..5 {
    tell i
}
```

Expected output:

```
1
2
3
4
5
```

18. Loop

What it demonstrates: Basic repeated execution.

```
set count = 1

loop {
    tell count
    count++

    if count > 3 {
        break
    }
}
```

Expected output:

```
1
2
3
```

19. Break

What it demonstrates: Stopping a loop early.

```
set i = 1

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

    tell i
    i++
}
```

Expected output:

```
1
2
3
4
```

20. Continue

What it demonstrates: Skipping an iteration.

```
set i = 0

while i < 5 {
    i++

    if i == 3 {
        continue
    }

    tell i
}
```

Expected output:

```
1
2
4
5
```

21. Nested Loops

What it demonstrates: Loop inside another loop.

```
for i in 1..3 {
    for j in 1..2 {
        tell i
        tell j
    }
}
```

22. Foreach

What it demonstrates: Iterating through a list.

```
set numbers = [10, 20, 30, 40]

foreach number in numbers {
    tell number
}
```

Expected output:

```
10
20
30
40
```

PART V — LISTS & OBJECTS

23. List Basics

What it demonstrates: Create and access list elements.

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

tell numbers[0]
tell numbers[1]
tell numbers[2]
```

Expected output:

```
10
20
30
```

24. List Processing

What it demonstrates: Combine foreach with a list.

```
set marks = [85, 90, 95]
set total = 0

foreach mark in marks {
    set total += mark
}

tell total
```

Expected output:

```
270
```

25. List Mutation

What it demonstrates: Modify an indexed list value.

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

set numbers[1] = 99

tell numbers
```

26. Object Basics

What it demonstrates: Create and access an object.

```
set user = {
    name: "Manideep",
    age: 21,
    active: true
}

tell user.name
tell user.age
tell user.active
```

Expected output:

```
Manideep
21
true
```

27. Nested Object

What it demonstrates: Access deeply nested values.

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

tell data.user.name
tell data.user.scores[0]
tell data.user.scores[2]
tell data.active
```

28. Nested Assignment

What it demonstrates: Modify nested object/list values.

```
set data = {
    user: {
```

```
        name: "Manideep",
        scores: [85, 90, 95]
    }
}

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

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

Expected output:

```
Daksh
99
```

29. 2D List

What it demonstrates: Modify a nested list index.

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

set matrix[0][1] = 99

tell matrix[0][1]
```

Expected output:

```
99
```

PART VI — FUNCTIONS

30. Simple Function

What it demonstrates: Declare and call a function.

```
function greet(name) {
    tell "Hello"
    tell name
}

greet("Manideep")
```

31. Function With Return

What it demonstrates: Return a computed value.

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

set result = add(10, 20)
tell result
```

Expected output:

```
30
```

32. Function With Multiple Parameters

What it demonstrates: Use several parameters.

```
function studentResult(name, marks) {
    set total = 0

    foreach mark in marks {
        set total += mark
```

```
    }

    return {
        name: name,
        total: total
    }
}

set result = studentResult(
    "Manideep",
    [80, 90, 95]
)

tell result.name
tell result.total
```

33. Recursion

What it demonstrates: A recursive factorial function.

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

    return n * factorial(n - 1)
}

tell factorial(5)
```

Expected output:

```
120
```

34. Function Returning a List

What it demonstrates: Return a collection from a function.

```
function makeNumbers() {
    return [10, 20, 30]
}

set numbers = makeNumbers()
tell numbers[1]
```

Expected output:

```
20
```

35. Function Returning an Object

What it demonstrates: Return structured data.

```
function makeUser(name, age) {
    return {
        name: name,
        age: age
    }
}

set user = makeUser("Manideep", 21)

tell user.name
tell user.age
```

Expected output:

```
Manideep
21
```

36. Function + Loop + Mutation

What it demonstrates: Combine functions, loops and list mutation.

```
function doubleNumbers(numbers) {
    set i = 0

    while i < list.length(numbers) {
        set numbers[i] = numbers[i] * 2
        i++
    }

    return numbers
}

import list

set values = [1, 2, 3]
set result = doubleNumbers(values)

tell result
```

PART VII — SCOPE & ERRORS

37. Block Scope

What it demonstrates: Outer variables are visible inside a nested block.

```
set name = "Manideep"

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

tell name

# message is not available here.
```

38. Function Scope

What it demonstrates: Function-local variables stay inside the function.

```
set name = "Outside"

function demo() {
    set message = "Inside"
    tell name
    tell message
}

demo()

tell name
# message is not available here.
```

39. Undefined Variable Error

What it demonstrates: Demonstrate a runtime error.

```
tell unknownValue
```

40. Division By Zero Error

What it demonstrates: Demonstrate invalid arithmetic.

```
set x = 10
```

```
tell x / 0
```

41. Invalid List Index Error

What it demonstrates: Demonstrate index validation.

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

42. Constant Modification Error

What it demonstrates: Demonstrate constant protection.

```
const pi = 3.14
set pi = 3.14159
```

PART VIII — INPUT & CONVERSION

43. Name Input

What it demonstrates: Read text from the user.

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

44. Numeric Input

What it demonstrates: Read and convert input.

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

tell age
```

45. Simple Calculator

What it demonstrates: Use input, conversion and arithmetic.

```
set a = integer(input("Enter first number: "))
set b = integer(input("Enter second number: "))

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

PART IX — STANDARD LIBRARY PROGRAMS

46. Math Module

What it demonstrates: Use math functions.

```
import math

tell math.abs(-10)
tell math.sqrt(144)
tell math.pow(2, 5)
tell math.floor(4.8)
tell math.ceil(4.2)
tell math.round(4.6)
tell math.max(10, 20, 5)
tell math.min(10, 20, 5)
tell math.pi
```

47. String Module

What it demonstrates: Clean and transform text.

```
import string

set name = "  manideep  "

set clean = string.trim(name)
set upperName = string.upper(clean)

tell clean
tell upperName
tell string.length(clean)
tell string.contains(clean, "deep")
```

48. List Module

What it demonstrates: Use list utilities.

```
import list

set numbers = [10, 20, 30]

tell list.length(numbers)
tell list.first(numbers)
tell list.last(numbers)
tell list.contains(numbers, 20)

list.add(numbers, 40)
tell numbers
```

49. Random Module

What it demonstrates: Generate random values.

```
import random

tell random.number()
tell random.integer(1, 10)

set choices = ["red", "green", "blue"]
tell random.choice(choices)
```

50. Time Module

What it demonstrates: Work with current date/time.

```
import time

set today = time.today()

tell today
tell time.now()
tell time.nowDateTime()
tell time.format(today, "YYYY-MM-DD")
tell time.dayName(today)
tell time.monthName(today)
tell time.yearNumber(today)
tell time.monthNumber(today)
tell time.dayNumber(today)
```

51. Time Calculations

What it demonstrates: Add/remove days and compare dates.

```
import time

set today = time.today()
set nextWeek = time.addDays(today, 7)
set previousDay = time.removeDays(today, 1)
```

```
tell nextWeek
tell previousDay
tell time.isAfter(nextWeek, today)
tell time.isBefore(previousDay, today)
```

52. System Module

What it demonstrates: Inspect runtime/system information.

```
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()
```

53. File Download

What it demonstrates: Create downloadable content in the browser runtime.

```
import file

file.download("daksh-example.txt", "Hello from Daksh!")
```

54. File Read

What it demonstrates: Open a local file through the browser file picker.

```
import file

set content = file.read()
tell content
```

55. File Write

What it demonstrates: Save content through the browser save dialog.

```
import file

file.write("Hello from Daksh!")
```

PART X — COMBINED PROGRAMS

56. Even/Odd Number Analyzer

What it demonstrates: Combine input, conversion and conditions.

```
set number = integer(
    input("Enter a number: ")
)

if number % 2 == 0 {
    tell "Even"
} else {
    tell "Odd"
}
```

57. Student Grade Calculator

What it demonstrates: Combine lists, foreach, functions and conditions.

```
import list
```

```
function gradeStudent(marks) {
    set total = 0

    foreach mark in marks {
        set total += mark
    }

    set average = total / list.length(marks)

    if average >= 90 {
        return "A"
    } else if average >= 75 {
        return "B"
    } else if average >= 60 {
        return "C"
    } else {
        return "F"
    }
}

set marks = [85, 90, 95]

tell gradeStudent(marks)
```

Expected output:

```
A
```

58. Shopping Bill

What it demonstrates: Combine objects, lists, loops and arithmetic.

```
set items = [
    { name: "Book", price: 250 },
    { name: "Pen", price: 20 },
    { name: "Bag", price: 800 }
]

set total = 0

foreach item in items {
    tell item.name
    set total += item.price
}

tell total
```

59. Number Search

What it demonstrates: Use loops and break.

```
set numbers = [10, 20, 30, 40, 50]
set target = 30
set found = false

foreach number in numbers {
    if number == target {
        set found = true
        break
    }
}

if found {
    tell "Found"
} else {
    tell "Not Found"
}
```

Expected output:

```
Found
```

60. Student Processing Program

What it demonstrates: A full Phase 0 integration example.

```
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

set student.average = 95
tell student.average

tell math.sqrt(144)
```

Expected output:

```
MANIDEEP
90
A
A+
95
12
```

PART XI — PRACTICE CHALLENGES

Try these yourself without looking at the previous solutions:

1. Build a calculator for two numbers.
2. Print numbers from 1 to 100 and skip multiples of 3.
3. Find the largest value in a list.
4. Find the sum and average of a list of marks.
5. Build a factorial program using recursion.
6. Create an object representing a student and update its grade.
7. Build a simple shopping bill using a list of objects.
8. Write a function that returns an object containing a student's name, average and grade.
9. Use the time module to calculate a date seven days from today.
10. Build a complete student-processing mini program using at least three standard-library modules.

Final Note

This is intentionally a separate example-program book. The main Daksh User Guide should be used for language reference and syntax; this PDF should be used for hands-on practice and demonstrations.

Daksh — Phase 0 Example Programs