MODULE 3: PROGRAMMING FOR BIOLOGICAL DATA

# Session 3: Logic and Control Flow

Building the "Auto-Verifier": If/Else and Loops

**Instructor: Dr. Gu Haogao**

January 12, 2026

# Recap: The Digital Lab So Far

## Session 1 & 2

We built **"Test Tubes"** (Variables).

We built **"Batches"** (Vectors).

We built **"Worksheets"** (Data Frames).

## The Missing Link

Right now, R is just a powerful calculator.

We need **Automation**.

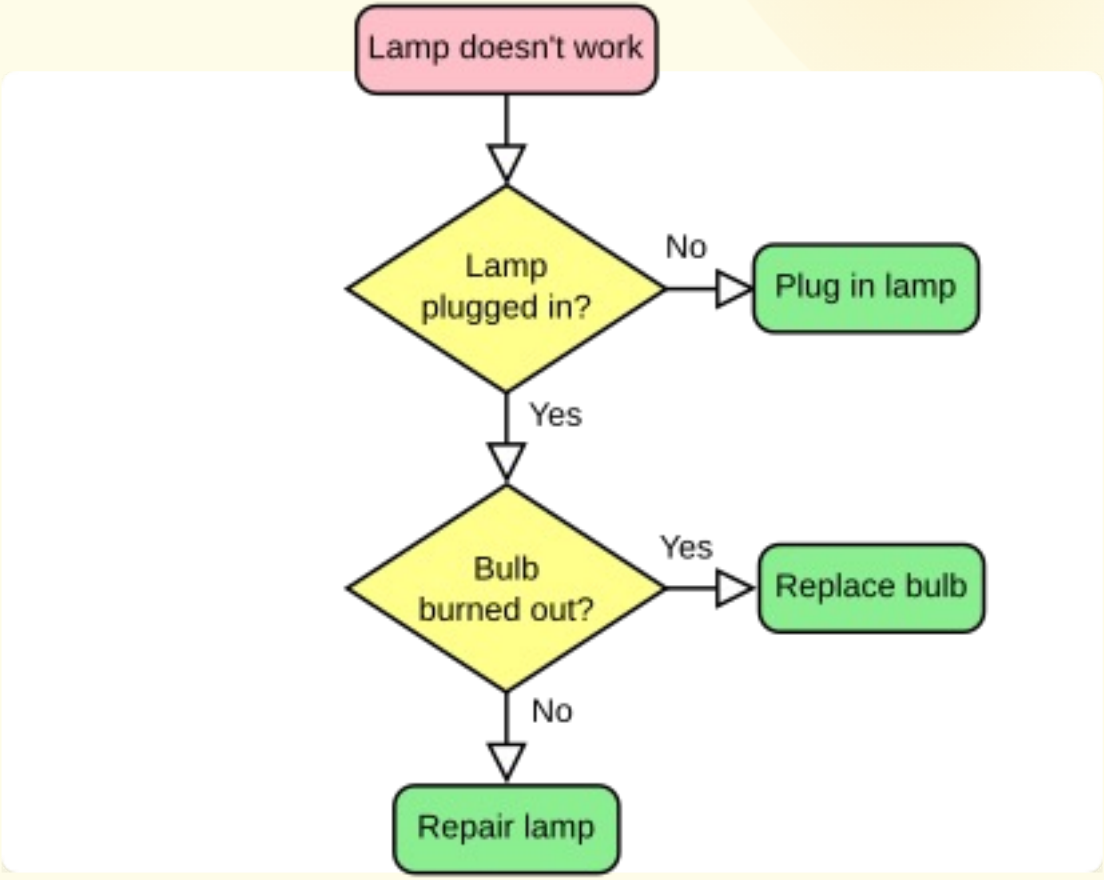
**"IF result > 6.5, THEN flag it."**

# The Concept: Logic in the Lab

Every time you sign off a result, you run a logical test in your brain.

- Is the QC within 2SD? (Yes/No)
- Is the patient HbA1c > 6.5%? (Yes/No)
- Is the sample hemolyzed? (Yes/No)

In programming, this is **Boolean Logic**.



# Boolean Logic: The Foundation

There are two reserved words in R that define truth.

## TRUE

The condition is met.

```
qc_pass <- TRUE
```

## FALSE

The condition is not met.

```
flag_error <- FALSE
```

*Note: Must be all CAPS. No quotes.*

# Asking Questions (Operators)

| Symbol | Meaning |
| --- | --- |
| == | Is equal to? (Double equals!) |
| != | Is NOT equal to? |
| < | Less than |
| > | Greater than |
| >= | Greater than or equal to |

### ⚠Common Mistake

= assigns a value.

== compares values.

# Practice: Asking R Questions

We give R numbers, it gives us Truth.

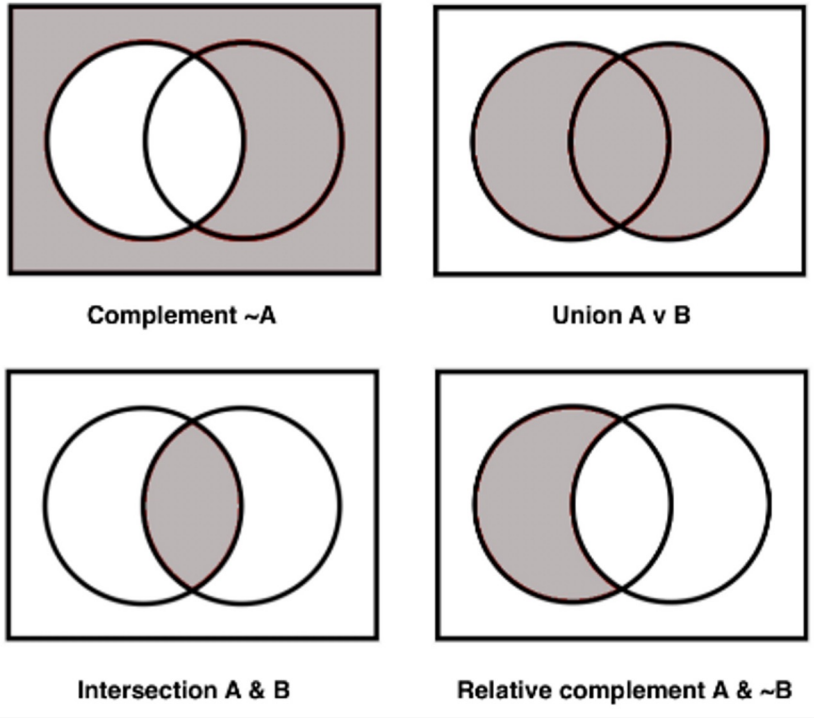
```
hb_a1c <- 7.2

# Ask: Is the patient diabetic?
hb_a1c > 6.5
# Output: [1] TRUE

# Ask: Is the patient normal?
hb_a1c < 5.7
# Output: [1] FALSE
```

# Logical Operators (Combining)

- & **(AND)**: Both must be TRUE.
  *"QC Pass AND Patient Negative"*
- | **(OR)**: At least one must be TRUE.
  *"Result Low OR Result High"*
- ! **(NOT)**: Reverse the logic.
  *"NOT Healthy"*

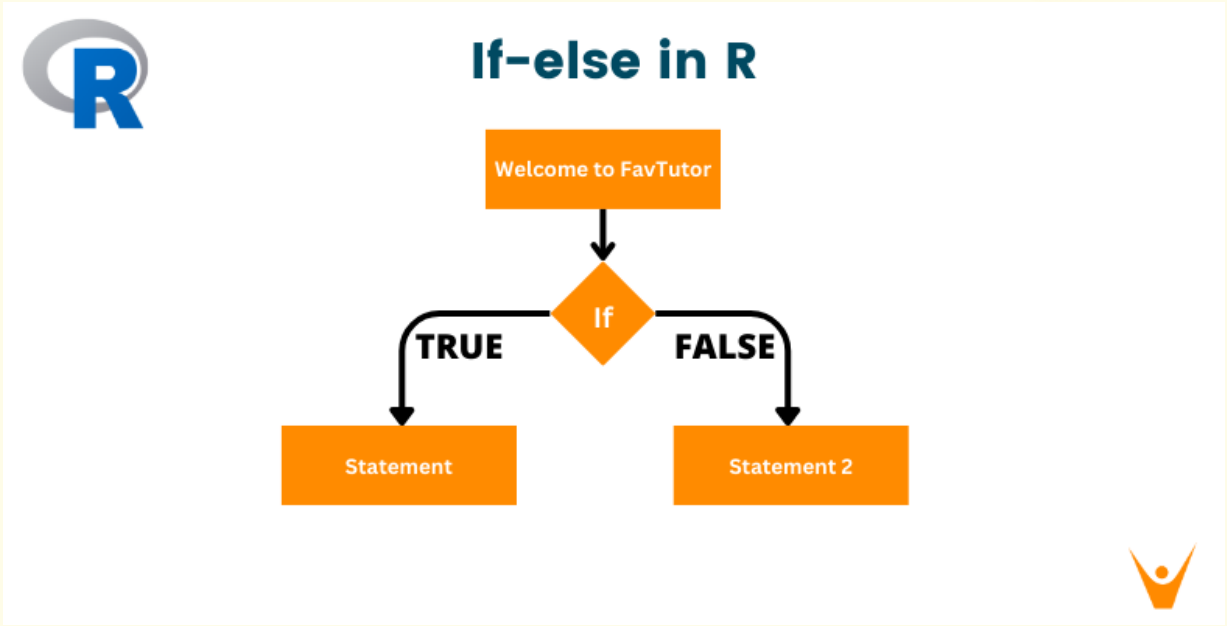


# The Conditional: if

The if statement is a gatekeeper. Code inside is a restricted area.

```
if (condition) {
   # Action to take if TRUE
}
```

R only opens the gate if the condition is TRUE.



# Example: The Safety Guard

```
potassium <- 7.5

if (potassium > 6.0) {
   print("CRITICAL ALERT: CALL WARD")
}
```

**Output:** [1] "CRITICAL ALERT: CALL WARD"

*If potassium was 4.0, nothing would happen. Silence.*

# The Alternative: else

The else block catches everything that didn't pass the first test.

```
if (potassium > 6.0) {
   print("Critical")
} else {
   print("Release Result")
}
```

This ensures every sample gets a classification.

# The Clinical Ladder: else if

Medicine isn't just High/Low. It's often Low, Normal, Borderline, High.

```
if (temp > 39) {
   print("High Fever")
} else if (temp > 37.5) {
   print("Mild Fever")
} else {
   print("Afebrile")
}
```

R checks in order. The first one that is TRUE wins.

# Challenge: HbA1c Rules

### Diagnostic Criteria (ADA)

**Normal:** < 5.7%

**Prediabetes:** 5.7% – 6.4%

**Diabetes:** >= 6.5%

*Keep this logic in mind. We will build an auto-verifier for this in the tutorial.*

# Part 2: Iteration (Loops)

Sometimes we need to pick up each sample, check it, tag it, and put it back.

**The "Conveyor Belt"**

Operator: for

# The for Loop Syntax

```
for (i in 1:5) {
   print(paste("Processing Sample #", i))
}
```

- 1:5: The sequence (the rack of tubes).
- i: The placeholder (the current tube in your hand).

# Looping through a Vector

```
batch_results <- c(5.5, 6.2, 4.8)

for (result in batch_results) {
   print(result * 10)
}
```

**Output:** 55, 62, 48

R grabs 5.5, multiplies it. Then grabs 6.2, multiplies it. Repeats until empty.

# Combining Logic and Loops

The Power Move: Putting an if inside a for.

## The Auto-Verifier

Iterate through the daily list.

Check rules for each patient.

Assign a flag.

## LIS Logic

This is the core logic of Laboratory Information Systems.

# The Auto-Verifier Pattern

```
results <- c(5.5, 7.2, 5.8)

# 1. Loop
for (val in results) {

   # 2. Logic
   if (val > 7.0) {
      print("Critical")
   } else {
      print("OK")
   }
}
```

Flow: Pick up sample -> Check value -> Print Label -> Next sample.

# Storing Results (The Empty Bucket)

Printing isn't enough. We need to save the data.

```
# Initialize empty vector (The Bucket)
flags <- c()

for (i in 1:3) {
   # Append result to the bucket
   flags[i] <- "Computed_Status"
}
```

Always set up an empty rack (vector) before starting the run.

# Functions: Reusable Protocols

Instead of writing the if/else ladder 100 times, write it once as a tool.

```
check_sugar <- function(val) {
   if (val > 6.5) {
      return("Diabetes")
   } else {
      return("Normal")
   }
}
```

You are building your own instrument named check_sugar().

# Applying Functions

```
patient_A <- 7.1

diagnosis <- check_sugar(patient_A)
```

# Vectorization vs. Loops

## Vectorization (The R Way)

df$Res > 5

Faster for math.

Best for simple calculations.

## Loops

for (i in list)

Essential for complex logic.

Necessary for file handling (e.g., reading 100 CSVs).

# Summary: Your Logic Toolkit

- **Boolean:** TRUE / FALSE
- **Operators:** ==, >, &, |
- **Conditional:** if (...) { ... } else { ... }
- **Loop:** for (x in list) { ... }

# Tutorial 3: The HbA1c Auto-Verifier

Goal: Build a script that mimics an LIS validation rule using real patient data.

1. Import Data
2. Write Diagnosis Loop
3. Generate Report

# Step 1: Load the Data

Load the 50-patient dataset from GitHub.

```
url <- "https://raw.githubusercontent.com/.../hba1c_results.csv"
df <- read.csv(url)

# Inspect
head(df)
```

**Check:** Ensure HbA1c_Level is numeric.

# Step 2: Initialize Storage

We need a place to put our diagnoses. Create an empty column.

```
# Create a new column filled with NA (placeholder)
df$Diagnosis <- NA

# Check structure
head(df)
```

# Step 3: The Logic (Mental Draft)

Write the logic for one number (e.g., 6.0) on paper or in a comment.

IF val >= 6.5 -> **"Diabetes"**

ELSE IF val >= 5.7 -> **"Prediabetes"**

ELSE -> **"Normal"**

# Step 4: The Loop (Structure)

Set up the loop to iterate through every row.

```
# Loop from 1 to the number of rows
for (i in 1:nrow(df)) {

   # Get the value for this specific row
   val <- df$HbA1c_Level[i]

   # (Logic goes here...)
}
```

# Step 5: Combining Logic & Loop

```
for (i in 1:nrow(df)) { val <- df$HbA1c_Level[i] if (val >= 6.5) { df$Diagnosis[i] <- "Diabetes" } else if (val >= 5.7) { df$Diagnosis[i] <- "Prediabetes" } else { df$Diagnosis[i] <- "Normal" } }
```

# Step 6: Verify Results

Check if the loop worked. Look at the top rows.

```
head(df, 10)
```

**Sanity Check:** Does Patient P-004 (11.2) say "Diabetes"? Does P-001 (5.2) say "Normal"?

# Step 7: Summary Stats

Count how many of each category we found.

```
# The table() function counts categories
counts <- table(df$Diagnosis)

print(counts)
```

# Step 8: Visualization (Bonus)

```
library(ggplot2)
ggplot(df, aes(x=Diagnosis, fill=Diagnosis)) + geom_bar() +
labs(title="Batch Summary: HbA1c")
```

# Step 9: Export

Save the fully verified worksheet.

```
write.csv(df, "Verified_HbA1c_Report.csv", row.names=FALSE)
```

**You have just built a functioning Auto-Verifier script!**

https://github.com/Koohoko/MSc_Module3_Programming_BioData_HKUSpacce_2026

README

# MSc Module 3: Programming for Biological Data (2026)

**Instructor:** Dr. Gu Haogao
**Institution:** SPH HKU / HKU SPACE **Date:** January 2026

## Course Overview

This repository contains the comprehensive preparation package for **Module 3**, tailored for Postgraduate Certificate in Bioinformatics for Medical Laboratory Technologists (MLTs). This plan integrates the "Code-as-Protocol" pedagogical approach.

## Repository Structure

- `lectures/` : Slides and R scripts used during the lecture.
- `tutorials/` : Interactive notebooks for hands-on practice.
- `data/` : Raw datasets used in exercises.
- `setup/` : Scripts to initialize the R environment.

## Quick Start

No software installation is required. We will use Google Colab for our "Dry Lab" sessions. Remember to **change runtime to R** while using colab (**Runtime -> Change runtime type -> R**).

- Session 1: Introduction to R (Part 1)
  - Lecture Script
  - Tutorial Notebook: Open in Colab
- Session 2: Introduction to R (Part 2) - Data Frames & I/O
  - Lecture Script
  - Tutorial Notebook: Open in Colab