# Introduction to Computer Programming

## - Week 3&4

- *Eng. Sylvain Manirakiza* -

Mon - Sep 29, 2025

# Quick Recap of Week 2

- Key Concepts:
  - Variables
  - Data types,
  - Assignment operations,
  - Introduced control structures.

# What is Variable?

- A Labeled storage location in memory that holds a value, which can change during program execution.
- It acts as a container for storing data like numbers, text, or Boolean values.

Example

- Variable Age as Integer

Age ← 25

# What is Data type?

- A data type in programming defines the kind of data a variable can store, such as numbers, text, or Boolean values.
- It helps the computer understand how to handle the stored data efficiently.

Example

**Common Data Types:**

- Integer (int) – *Whole numbers (e.g., 10, -3).*
- Float – *Decimal numbers (e.g., 3.14, -0.99).*
- String (str) – *Text values (e.g., "Hello").*
- Boolean (bool) – *True/False values (e.g., True, False).*

# Assignment operation

- An assignment operation is the process of storing a value in a variable using the assignment operator (← ).

Example

Variable **Age** ← 25

# Control structures

- Instructions that determine the flow of execution in a program.
- They help make decisions, repeat tasks, or execute statements in order.

Example

```
If age >= 18 Then
  Write "You are an adult."
Else
  Write "You are a minor."
End If
```

# Chap 2. Variables and Operations | Control Structures

# 🎯Today's Objectives

| | |
|---|---|
| **Objective 1** | • Understand different types of control structures |
| **Objective 2** | • Learn about conditional and iterative structures. |
| **Objective 3** | • Implement decision-making using If-Else and Switch statements. |
| **Objective 4** | • Explore loops for repeated execution. |

# What are Control Structures?

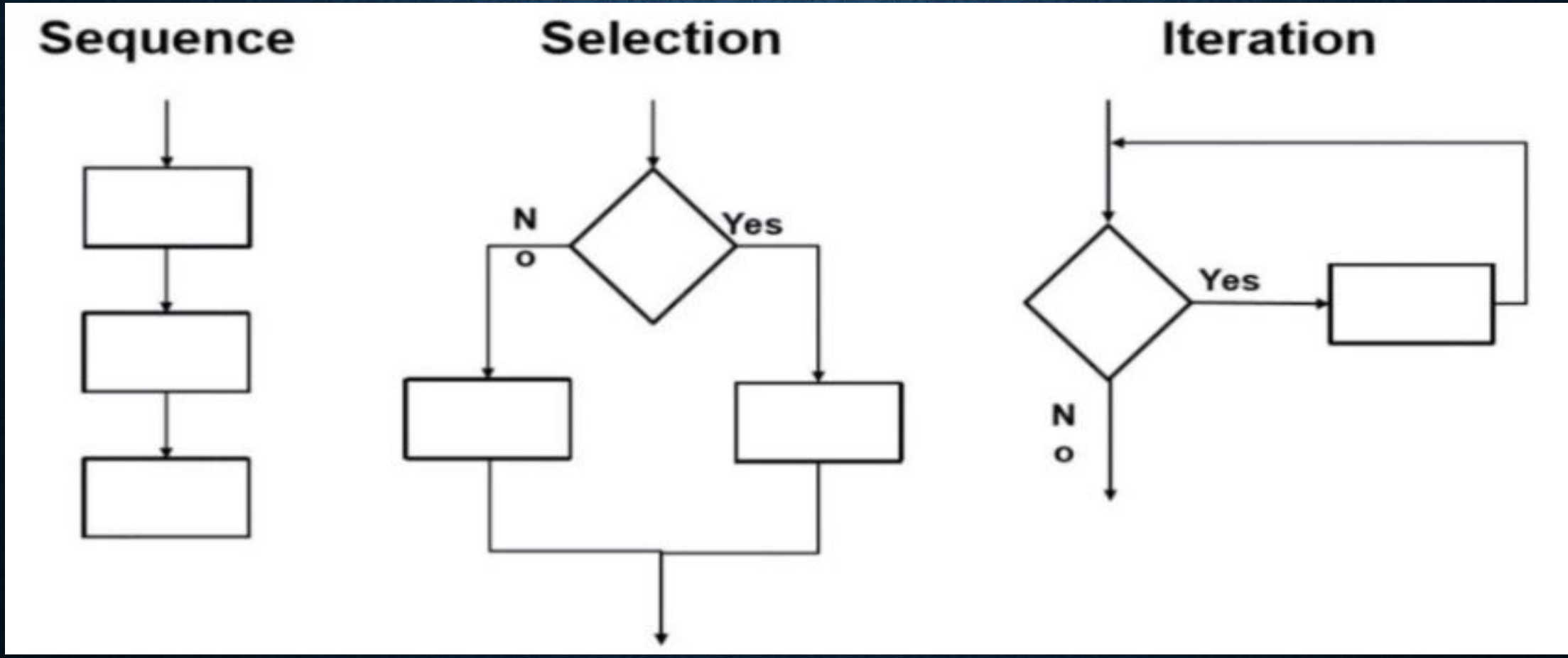
## Definition

Control structures guide the sequence in which instructions execute in a program

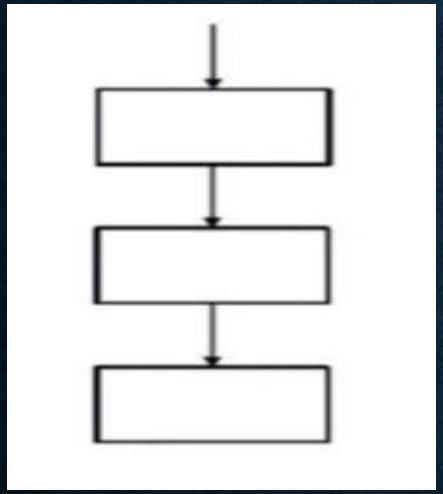
## Three main types

1. Sequential Execution
2. Selection (Decision-making)
3. Iteration (Loops)

# What are Control Structures?



# 1. Sequence

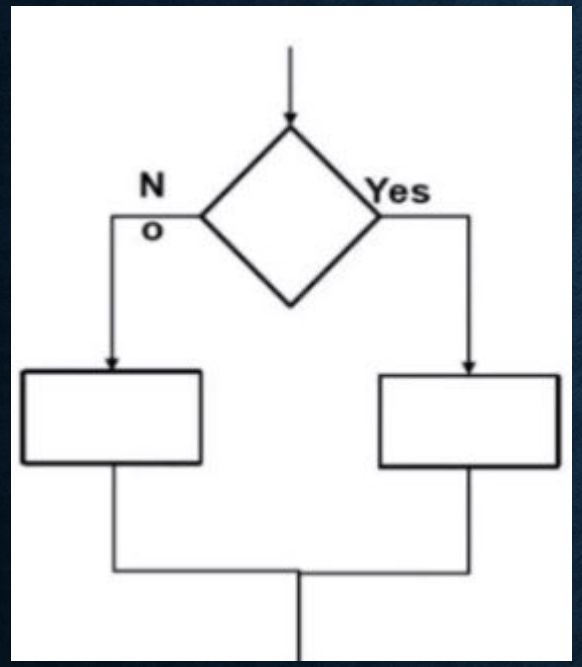


Instructions run from top to bottom

**Example:** Add two numbers

```
Variables num1, num2, sum as Integer
Start
  Write "Enter two numbers: "
  Read num1, num2
  sum ← num1 + num2
  Write "Sum is:", sum
End
```

# 2. Selection /Decision making

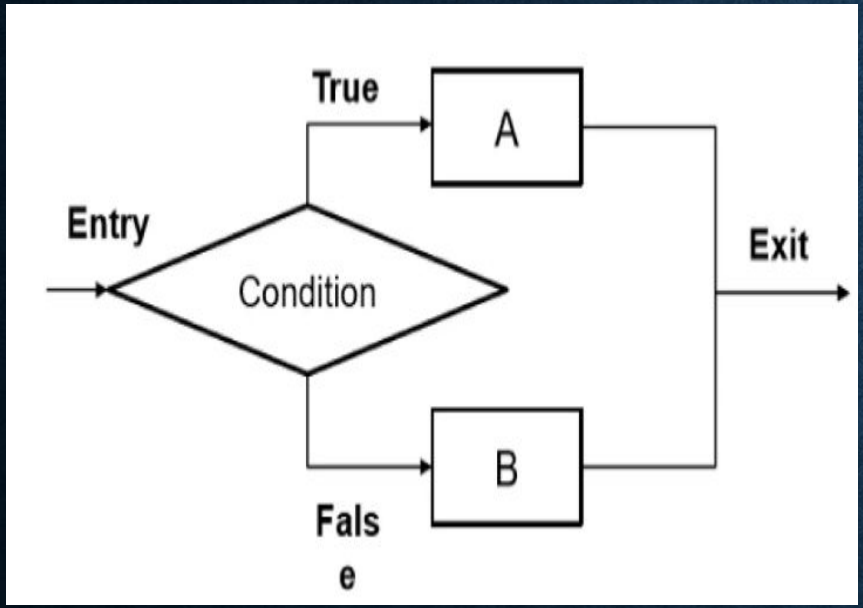



Choosing between multiple execution paths

3 Components of the structure:

1. A condition to be tested
2. The statement to be performed if the condition is satisfied (Process A)
3. The statement to be performed if the conditions in not satisfied (Process B)

# 2.1. Selection - If -Else




Choosing between multiple execution paths

**Example:**

```
Start
  Write "Enter your age:"
  Read age
  If age >= 18 Then
    Write "You are eligible to vote."
  Else
    Write "You are not eligible to vote."
  End If
End
```

# 2.2.1 Selection - Compound conditions

When more than one condition needs to be checked together.

**Logical Operators:**

- **AND** (`&&` or `AND`): Both conditions must be `True`.
- **OR** (`||` or `OR`): At least one condition must be `True`.
- **NOT** (`!` or `NOT`): Reverses the condition.

**Example: -AND**

```
Start
  Write "Enter your age:"
  Read age
  Write "Enter your citizenship status (yes/no):"
  Read citizen
  If age >= 18 AND citizen = "yes" Then
    Write "You are eligible to vote."
  Else
    Write "You are not eligible to vote."
  End If
End
```

# 2.2. 2 Selection - Compound conditions

When more than one condition needs to be checked together.

**Logical Operators:**

- **AND** (`&&` or `AND`): Both conditions must be `True`.
- **OR** (`||` or `OR`): At least one condition must be `True`.
- **NOT** (`!` or `NOT`): Reverses the condition.

Example: **-OR**

```
Start
  Write "Enter your age:"
  Read age
  If age < 18 OR age > 60 Then
    Write "You are eligible for a discount."
  Else
    Write "No discount available."
  End If
End
```

# 2.2.3 Selection - Compound conditions

When more than one condition needs to be checked together.

**Logical Operators:**

- **AND** (`&&` or `AND`): Both conditions must be `True`.
- **OR** (`||` or `OR`): At least one condition must be `True`.
- **NOT** (`!` or `NOT`): Reverses the condition.

**Example: -NOT**

```
Start
  Write "Enter a number:"
  Read num
  If NOT (num > 0) Then
    Write "The number is not positive."
  End If
End
```

# 2.3. Selection - Nested If

When one If statement is placed inside another.

**Example:**

```
Start
  Write "Enter a number:"
  Read num
  If num > 0 Then
    If num % 2 == 0 Then
      Write "The number is positive and even."
    Else
      Write "The number is positive and odd."
    End If
  Else
    Write "The number is not positive."
  End If
End
```

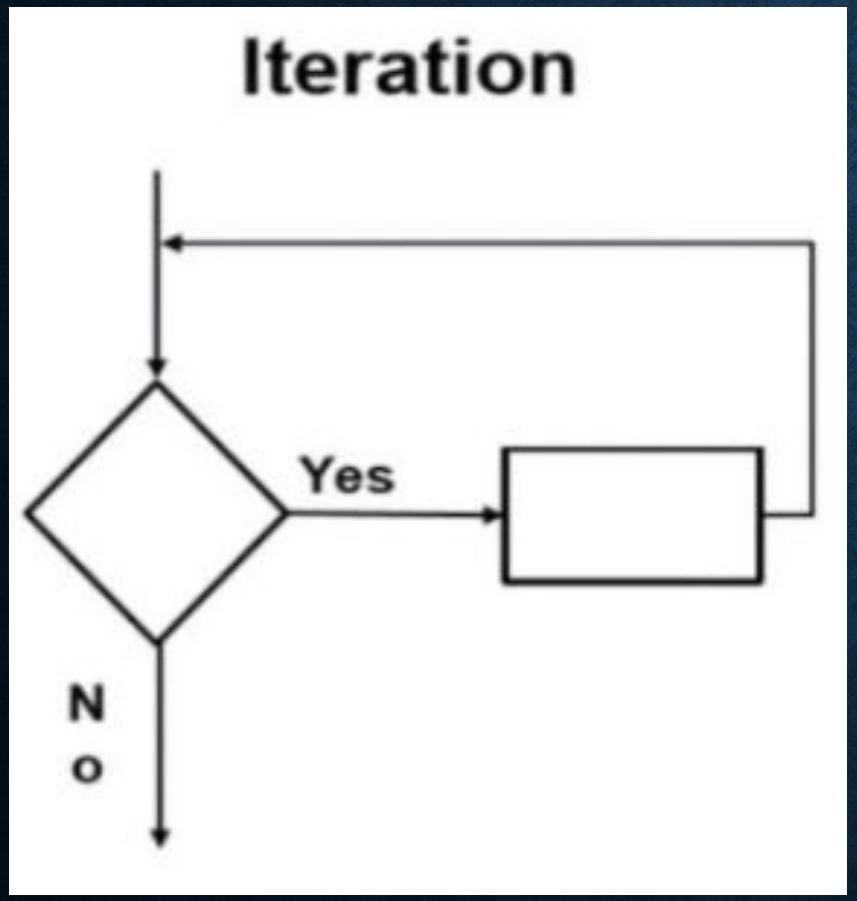
# 2.4. Selection - Switch Case

Used when multiple conditions depend on a single variable.

**Example:**

```
Start
 Write "Enter a day number (1-7):"
 Read day
 Switch day
   Case 1:Write "Monday"
   Case 2:Write "Tuesday"
   Case 3:Write "Wednesday"
   Case 4:Write "Thursday"
   Case 5:Write "Friday"
   Case 6:Write "Saturday"
   Case 7:Write "Sunday"
   Default:Write "Invalid Day"
 End Switch
End
```

# 3. Iteration/Loops




Iteration structures allow a program to execute a block of code multiple times.

# 3.1. Iteration - For Loop

Executes a block of code a fixed number of times.

Example: **Counting from 1 to 5**

```
Start
  For i ← 1 to 5 Do
    Write i
  End For
End
```

- This program prints numbers from 1 to 5
- The loop repeats until the end value is reached

# 3.2. Iteration - While Loop(Pre-Tested Loop)

Executes a block of code **while** a condition is `True`

Example 1: **Display the word "Hello" 5 Times**

```
Start
Declare i as Integer
i ← 1
While i ≤ 5 Do
   Write "Hello ", i
   i ← i + 1
End While
End
```

- This program prints the word"Hello" Five times, but it only executes while the condition `i <= 5` holds true.
-

# 3.2. Iteration - While Loop(Pre-Tested Loop)

**Syntax**

While condition is true Do

//statements

End While

Example 2: **ATM Machine**

```
Start
Declare continue as String
continue ← "yes"
While continue = "yes" Do
   Write "Withdraw money"
   Write "Do you want another transaction? (yes/no)"
   Read continue
End While
End
```

- The ATM allows withdrawals until the user chooses to stop.

# 3.3. Iteration - Do-While Loop(Post-Tested Loop)

Executes a block of code at least once, then repeats while a condition is `True`

Example 1: **Displaying 'Hello' word 5 times**

```
Start
Declare i as Integer
i ← 1
Do
   Write "Hello ", i
   i ← i + 1
While i ≤ 5
End
```

- This program ensures that at least 'Hello' word is printed once even if the i violates the condition

# 3.3. Iteration - Do-While Loop(Post-Tested Loop)

**Syntax**

Do

// statements

While condition is true

**Example 2: Game Playing**

```
Start
Declare playAgain as String
Do
    Write "Play game round"
    Write "Do you want to play again? (yes/no)"
    Read playAgain
While playAgain = "yes"
End
```

- A player plays at least one round of a game and then decides if they want to play again.

# 3.4. Iteration - Break and Continue Statements

1. Stops the loop immediately

**Example: Using Break**

```
Start
  For i ← 1 to 10 Do
    If i = 5 Then
      Break
    End If
    Write i
  End For
End
```

- This program prints numbers 1 to 4, but the loop stops when `i = 5`.

# 3.5. Iteration - Break and Continue Statements

1. **Skips** the current iteration and moves to the next

**Example: Using Continue**

```
Start
  For i ← 1 to 5 Do
    If i = 3 Then
      Continue
    End If
    Write i
  End For
End
```

- This program prints 1, 2, 4, 5, skipping `3`.
- It is useful for skipping unnecessary iterations (e.g., ignoring invalid inputs).

# 3. Iteration/Loops - Summary

Loops allow efficient repetition of tasks.

- **For loops** are best when the number of iterations is known.
- **While loops** are useful when the condition determines the repetitions.
- **Do-while loops** ensure execution happens at least once.
- **Break and Continue** help control loop execution.

# End

Group Exercises on **While** and **Do-While Loops.** **Click Here** to access them

Group Assignment(***Click Here*** *to access it*) to be Presented and Submitted by the next class. - Same way we did the previous presentations