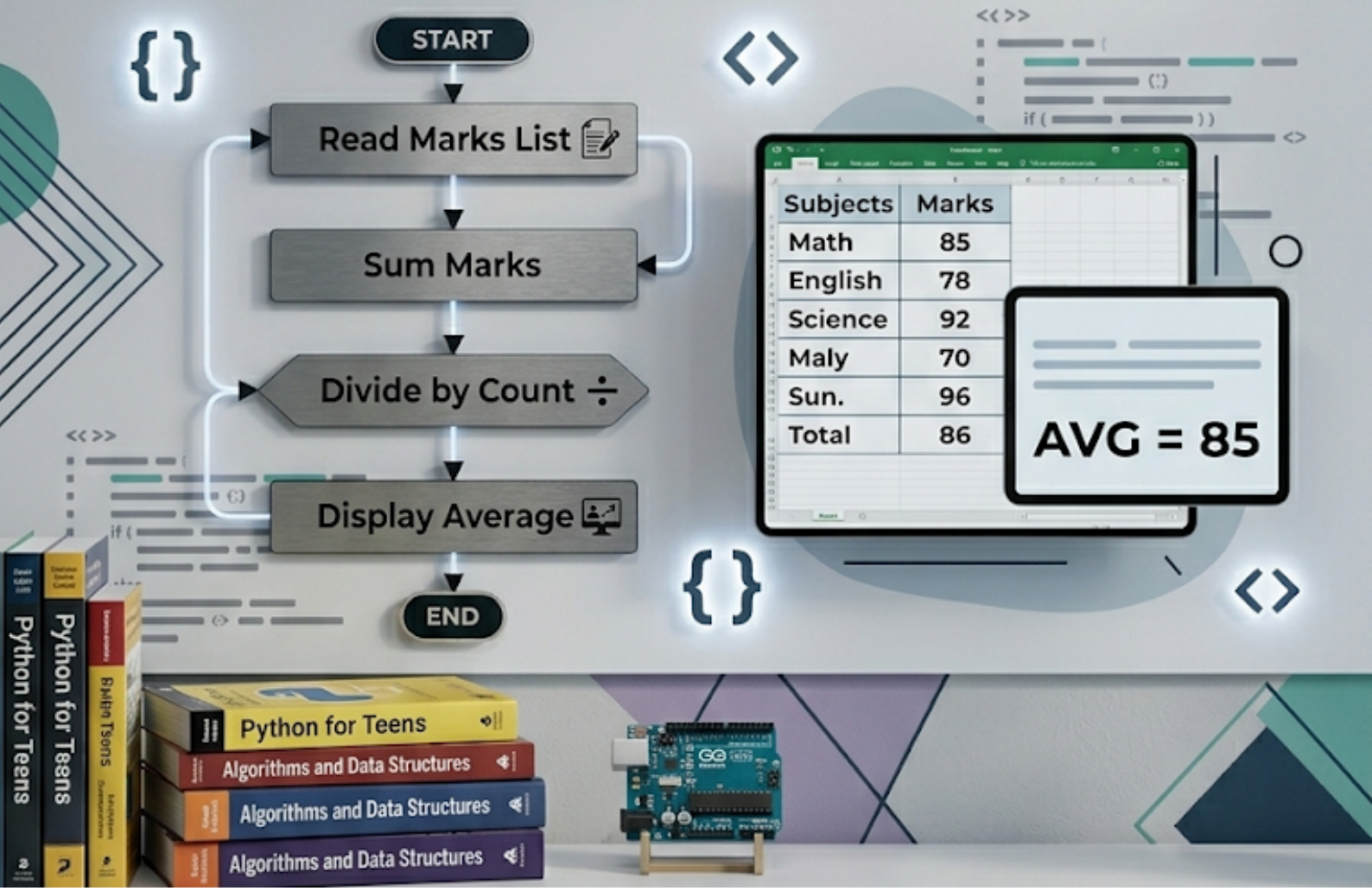
Windroid Academy

GRADE 9 PROGRAMMING LESSON

Topic: Calculating Student Averages



Unit: Control Structures, Flowcharts, and Arrays in Scratch
Welcome to the enhanced study guide for this unit. This lesson will teach you how to think like a computer programmer by breaking down complex problems, controlling program flow using logic, and managing data efficiently.

1. Selection Control Structures

In programming, **selection** is how a computer makes decisions based on conditions.

A. Simple Selection (Review)

A simple selection checks **one condition** and chooses between **two pathways** (True or False).

- **Real-world Example:** Checking if you have passed an exam. If your mark is 50 or above, you pass; otherwise, you fail.

B. Multi-Condition Selection (Selection Out of Many Options)

When a problem has more than two possible outcomes, we use multi-condition selection. Instead of a single path, the program evaluates conditions sequentially until it finds a true one.

Enhanced Real-World Example: Smart Ticket Pricing System
Imagine an automated ticketing counter at a botanical garden. The ticket price changes based on age categories:

- Age under 5: **Free**
- Age 5 to 12: **Half Price**
- Age 13 to 60: **Full Price**
- Age over 60: **Senior Discount Price**

Instead of checking just one condition, the system checks multiple conditions step-by-step using a flowchart.

2. Repetition Control Structures (Loops)

Repetition, or looping, allows a program to execute a block of code multiple times without rewriting it. Every loop requires a condition to determine when it starts and when it stops.
In Scratch, there are **three primary types** of repetition control structures:

This example: Until the basket is empty you have to fold the clothes

Repetition Type	How It Works	Scratch Block Name	Practical Example
Definite Repetition	Repeats an action for a **fixed, specific number** of times.	repeat (10)	Printing exactly 12 pages of a document.
Conditional Repetition	Repeats an action **until a specific condition** becomes true.	repeat until < >	Pumping water into a tank *until* the water level sensor detects it is full.
Infinite Repetition	Repeats an action **endlessly** while the program runs.	forever	A security camera constantly scanning a room for motion.

Enhanced Flowchart: Smart Water Pump System
Notice how the condition **"Is the tank full?"** controls the loop. As long as the answer is **No**, the pump continues to run. The moment it becomes **Yes**, the loop breaks, and the pump stops.

3. Nested Repetition (Loops Inside Loops)

A **nested loop** occurs when a repetition control structure is placed entirely inside another repetition control structure. For every single turn of the outer loop, the inner loop runs its entire cycle.

Classic Concept: Clockwork Logic
Think of how a digital clock tracks time:

- **Outer Loop:** Changes the **Hour** (repeats 24 times a day).
- **Inner Loop:** Changes the **Minute** (repeats 60 times for *every single* hour).

Visual Code Example: Drawing a Brick Wall Grid
If you want a Scratch sprite to draw a grid of square tiles, you use nested loops:

- **Inner Loop:** Draws 4 sides to complete a single square block.
- **Outer Loop:** Moves the sprite forward to repeat that square block 5 times in a row.

The Foundation: Algorithms and Flowcharts
Before writing code, programmers break problems down.

- **Algorithm:** A finite set of clear, step-by-step text instructions to solve a problem.
- **Flowchart:** A visual representation of the algorithm using standard shapes (Ovals for Start/End, Parallelograms for Input/Output, Rectangles for Processing, and Diamonds for Decisions).

Multi-Condition Selection Structures
A simple selection chooses between two paths. A **multi-condition selection** checks multiple possibilities in a sequence until it finds a true condition.
Scenario: A business automation system calculating the bulk discount for "GrainMuse Instant Rice" distributed to local university canteens

The Algorithm:

1. **Begin**
2. **Input** the number of rice packets ordered.
3. **If** the order is greater than 500 packets, **Then** set discount to 20%.
4. **Else If** the order is greater than 200 packets, **Then** set discount to 10%.
5. **Else If** the order is greater than 50 packets, **Then** set discount to 5%.
6. **Else** set discount to 0%.
7. **Output** the final discount amount.
8. **End**

Scratch Implementation: This is built using multiple `if... then... else` blocks slotted inside one another to check the order quantity step-by-step.

Repetition Control Structures (Loops)

Repetition structures allow a block of code to run multiple times based on a specific condition, which is essential for managing data like inventory.
Scenario: A web dashboard that automatically restocks ribbon flower craft supplies until the inventory reaches the maximum capacity of 100 units.

The Algorithm (Conditional Repetition):

1. **Begin**
2. **Input** current stock level.
3. **While** current stock is less than 100, **Do:**
4. Add 1 to current stock.
5. **Output** "Restocking... Current stock is: [stock level]"
6. **End While**
7. **Output** "Inventory Full."
8. **End**

Scratch Implementation:

This translates directly to the `repeat until <stock = 100>` block in Scratch, automatically looping the addition process until the condition is met.

Nested Repetition (Loops Inside Loops)

A **nested loop** occurs when you place one repetition structure entirely inside another. The inner loop must finish its entire cycle for every single step of the outer loop.
Scenario: A real-time revenue tracking system calculating daily sales totals for a full 7-day week.

The Algorithm:

1. **Begin**
2. Set Day_Count to 1.
3. **While** Day_Count is less than or equal to 7, **Do:** (Outer Loop)
4. Set Daily_Total to 0.
5. **Input** "Are there more sales today? (Yes/No)"
6. **While** answer is "Yes", **Do:** (Inner Loop)
7. **Input** sale amount.
8. Add sale amount to Daily_Total.
9. **Input** "Are there more sales today? (Yes/No)"
10. **End While**
11. **Output** Daily_Total for the day.
12. Add 1 to Day_Count.
13. **End While**
14. **End**

Managing Data with Arrays (Lists)
When you need to store multiple pieces of related data, using single variables (like sale1, sale2, sale3) becomes impossible to manage. An **Array** (called a **List** in Scratch) stores multiple items under one single name, using an index number to track positions.
Scenario: A business owner analyzing a 3-month sales slump wants to store the monthly revenue data to calculate the overall average.
The Algorithm (Using an Array):

1. **Begin**
2. Create an empty array named Monthly_Revenue.
3. Set Total_Sum to 0.
4. **Repeat** 3 times:
5. **Input** revenue for the month.
6. Add the inputted revenue to the Monthly_Revenue array.
7. **End Repeat**
8. Set Index to 1.
9. **While** Index is less than or equal to 3, **Do:**
10. Add the value at Monthly_Revenue[Index] to Total_Sum.
11. Add 1 to Index.
12. **End While**
13. Calculate Average = Total_Sum / 3.
14. **Output** Average.
15. **End**

Scratch Implementation:

You utilize the Make a List feature in the Data/Variables category. By combining a list with a repeat block, you can cycle through all stored data points in seconds, making your code efficient and professional.

4. Programming with Arrays (Lists in Scratch)

Arrays are one of the most powerful tools in coding! Before we start, here is a quick translation: **in Scratch, an array is called a "List."** Think of a regular variable as a single box that can only hold one item at a time. An array (or List) is like a filing cabinet with numbered drawers. You can store many different items in it, add new drawers, or take things out, all perfectly organized by their drawer number.

Here is a simple, interactive lesson to help you build your first array in Scratch.

🎒 Project: The Magic Backpack

Goal: We are going to code the Scratch Cat to use a "Backpack" list. You will learn how to add items, read what is inside, and empty the backpack.

Step 1: Create Your Array (The List)

First, we need to create the actual filing cabinet to store our data.

1. Open a new Scratch project.
2. On the left side, click the orange **Variables** category.
3. Click the **Make a List** button.
4. Type the name **Backpack** and click OK.

Notice that a grey box titled "Backpack" now appears on your stage. This is your array! It starts completely empty.

Step 2: Empty the Backpack on Start

It is good practice to clear your array every time you restart your game so you don't end up with old data.

1. Go to the yellow **Events** category and drag out the `when [green flag] clicked` block.
2. Go to the orange **Variables** category.
3. Drag the `delete (all) of [Backpack]` block and snap it under the green flag block.

Step 3: Add Items to the Array

Now let's program the spacebar to add a random item to our backpack.

1. Go to **Events** and drag out the `when [space] key pressed` block.
2. Go to **Variables** and drag out the `add [thing] to [Backpack]` block. Snap it under the spacebar event.
3. Change the word "thing" to something you'd put in a backpack, like "Apple" or "Flashlight".

Test it out: Press the spacebar a few times. Watch the grey box on your screen. You will see your items being added to the array, each with its own number (its "index").

Step 4: Read from the Array

Arrays are only useful if we can see what is inside them. Let's make the Scratch Cat tell us what the first item in the backpack is when we click on him.

1. Go to **Events** and drag out the `when this sprite clicked` block.
2. Go to the purple **Looks** category and drag out the `say [Hello!] for (2) seconds` block. Snap it under the click event.
3. Go back to **Variables**. Find the block that says `item (1) of [Backpack]`.
4. Drag that block and drop it directly over the word "Hello!" inside the `say` block.

Test it out: Click the Scratch Cat. If your backpack has items in it, he will say whatever is sitting in spot #1 of your array.

The Problem with Single Variables

A variable can only hold **one piece of data at a time**. If you want to track the test scores of 5 students, you would need 5 separate variables: `score1`, `score2`, `score3`, `score4`, and `score5`.
If you have 100 students, your code becomes messy, massive, and incredibly difficult to manage.

The Solution: Arrays
An **Array** (called a **List** in Scratch) is a structured container that can store multiple items under a **single variable name**. Each item in the list has an index number (a position) starting from 1.

Why Arrays + Loops are Powerful
By combining an array with a repetition loop, you can process thousands of data points with just a few lines of code.

Enhanced Example: Automated Stock Inventory Alert
Imagine an online retail store. The items are stored in a list called `Stock_Count`. A `repeat` loop can automatically check every item position in that list. If any item's stock count drops below 5, the program instantly triggers a "Low Stock Alert!" message.

5. Problem Analysis & Decomposition
Before writing a single line of code in Scratch, complex problems should be broken down into four foundational logical steps: **Input, Storage, Process, and Output.**
Let’s analyze the problem of calculating the average grade of a class:

- **Step 1: Input & Storage**
  - Create a list named `Grades_List`.
  - Prompt the user to enter marks and add them to the list using a loop.
- **Step 2: Processing (Summation)**
  - Create a variable named `Total_Sum`.

    - Use a loop to iterate through the list, adding each grade to `Total_Sum`.
- **Step 3: Processing (Calculation)**
    - Create a variable named `Average`.
    - Divide `Total_Sum` by the total number of items in the list.
- **Step 4: Output**
    - Have the Scratch sprite say: *"The class average is: [Average]"*.