Simulating flight forces with the Micro:bit	Grade 6, Stand D Flight

Lesson Plan

Coding Tool	Makecode and Micro:bits
Cross-curricular	Coding and Flight

Big Ideas

Students will use their coding skills (understanding of variables and conditional statements) to code a simple flight simulator that models the basic forces of flight: thrust, drag, lift and weight.

Students will experiment with their simulator by changing the weight and thrust of their plane to see how that impacts whether the plane goes up, drops or stays level.

Specific Expectations

A2.1 write and execute code in investigations and when modelling concepts, with a focus on obtaining input in different ways for a variety of purposes

D2.2 describe the relationships between the four forces of flight – lift, weight, thrust, and drag – that make flight possible

D2.4 describe ways in which the four forces of flight can be altered

Materials

- Chromebook or laptop for each student
- Optional: microbits (the code can be designed and tested using only the micro:bit website, however the learning is more impactful if you have physical micro:bits)

Introduction

In this lesson, we will explore the four forces of flight and see how they interact to keep an aircraft in the air. These forces work in pairs that oppose each other, and understanding their balance helps us explain how airplanes climb, descend, speed up, or slow down.

Lift and weight are opposite forces that determine whether an airplane gains or loses elevation.

- Lift: The upward force created by the movement of air around the wings. Lift must be greater than weight for an airplane to rise.
- Weight: The downward force caused by gravity pulling the airplane toward the Earth.

Drag and thrust are another pair of opposing forces that influence the airplane's speed, and therefore affect lift as well.

- Drag: The force that resists motion as the airplane moves through the air.
- Thrust: The forward-moving force produced by engines or propellers that pushes the airplane ahead.

Throughout the activity, students will see how all four forces must stay in balance for stable flight.

In the coding portion of this lesson, we will focus on two essential coding concepts:

- Variables: Values that can change during a program and can be used to store information such as speed, weight and lift.
- Conditional Statements: "If...then..." rules that allow the program to make decisions. For example, showing the plane rising if lift is greater than the force of the weight.

Students will use these concepts to simulate the forces of flight and observe how changing one variable affects the others, reinforcing both the science and the coding skills.

Action

Have each student visit the *Micro:bit coding website*, "https://makecode.microbit.org/", and start a *new project*.

Step 1: Identifying and Creating Our Variables

Explain that students will be building a simple flight simulator that models the four important forces involved in flight.

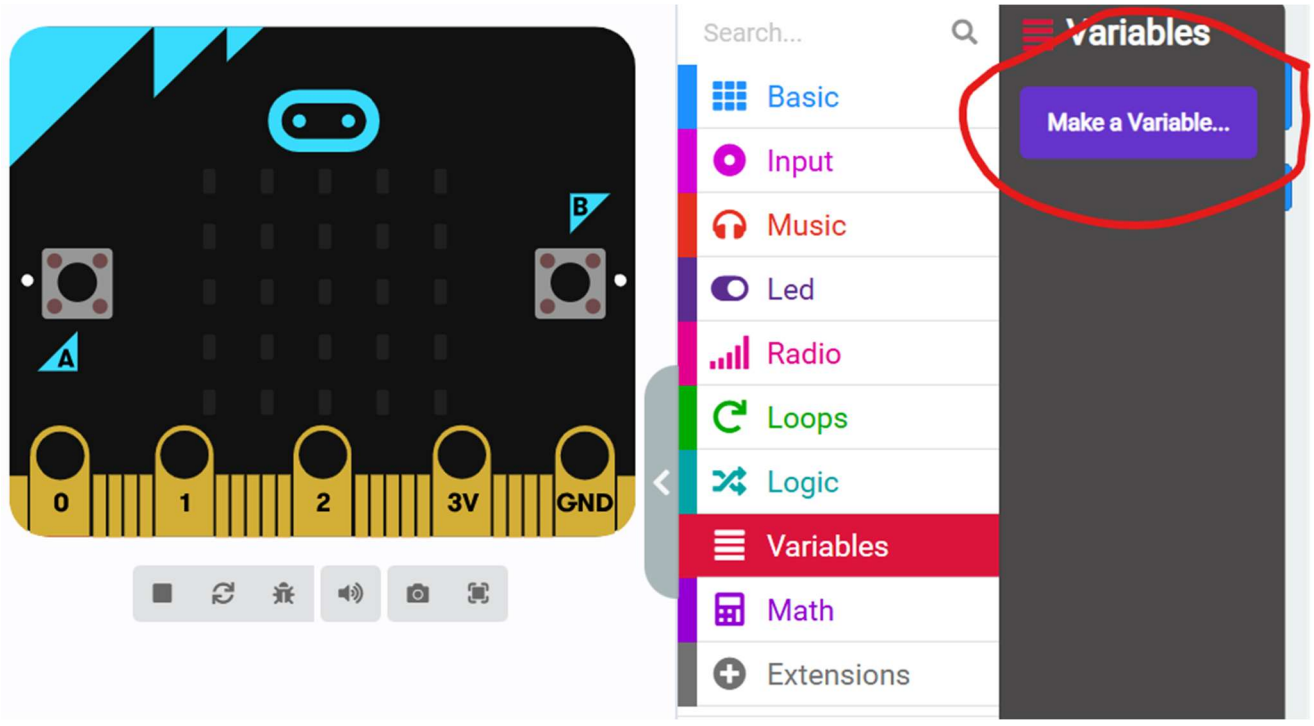
If needed, briefly review the four opposing forces — weight and lift, and thrust and drag (explanation found in the introduction section).

To build this simulator, students will create variables that will change throughout the program and determine whether the plane flies upward, descends, or stays level.

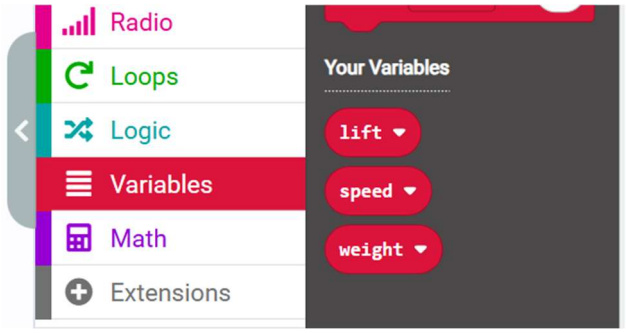
The variables students will code for are *lift*, *weight*, and *speed*.

Note: For simplicity, "speed" will represent both thrust and drag. Increasing speed implies thrust, while decreasing speed implies drag.

Click the *red Variables tab* and select *make a variable*. You will need to create three variables: "speed" (this will represent both *thrust* and *drag*), "weight", and "lift".



Once the variables are created, you should see them listed in the *Variables menu* under "Your Variables".



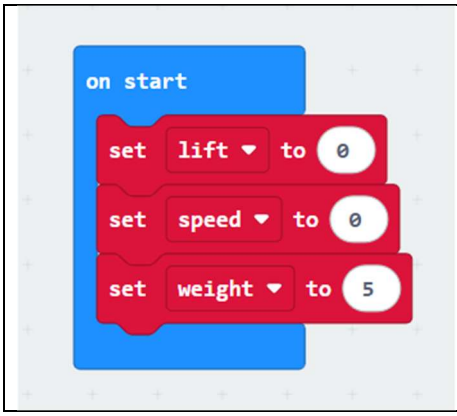
Step 2: Setting starting values

Explain that we need to set our variables at the beginning of the program; otherwise, the computer will not know their values. Computers only follow the instructions we give them, so clear starting values are essential.

If there is not already one in the workspace, go to the *blue Basic tab* and drag an "on start" block onto the workspace.

Next, drag in three *"set variable to"* blocks and place them inside the "on start" block—one for each of our variables: *lift*, *weight*, and *speed*.

- Set speed to 0
- Set lift to 0
- Set weight to 5 (an arbitrary value; you can assign a unit such as "5 kg model plane" if preferred)



Note to students: This means that when our code starts, the plane is still because the speed is zero, the plane has weight, and the plane has no lift since it has no power or thrust yet.

Step 3: Coding the main flight forces simulator

If a "forever" block is not already in your workspace, drag a *"forever" block* from the *blue Basic tab* into your workspace.

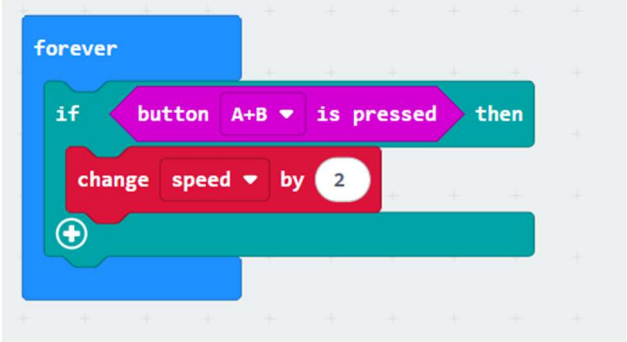
3A: Adding Thrust and Drag

Begin by placing an *"if" block* from the *teal Logic tab* inside the "forever" loop. This block will allow us to code what happens when thrust is added, and what happens when thrust is removed.

Next, add the "if *Button is pressed*" hexagonal block, also found in the *teal Logic tab*. Use the drop-down menu on the block to select "A+B".

Finally, drag in a *"change variable by"* block from the *red Variables tab*, select "speed" as the variable, and change the value to *2*. This will increase the plane's speed whenever the A+B buttons are pressed, simulating added thrust.

Your code should, look like this:

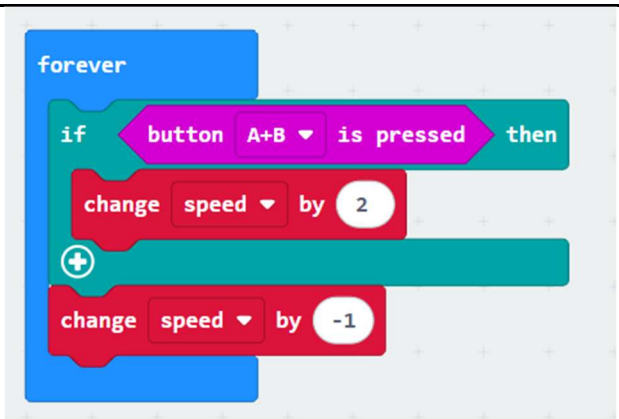


This means that when we *hold down buttons A and B*, we are adding *speed* to the plane. This represents the *thrust force*, which increases the plane's speed and contributes to *lift* (we will code that interaction shortly).

To add the effects of *drag* on our plane, place a "change variable by" block directly *below* the "if" conditional statement. Make sure to select "speed" again as the variable and change the value to *–1*.

This simulates drag by slowly decreasing speed whenever thrust is not being added.

Our *forever* loop should now look like this:



This means that *drag is always acting* on our plane, causing the *speed* to decrease, which in turn affects the plane's *lift*.

This helps demonstrate to students why planes require a continuous *thrust input* to remain in the air.

3B: Preventing negative speed

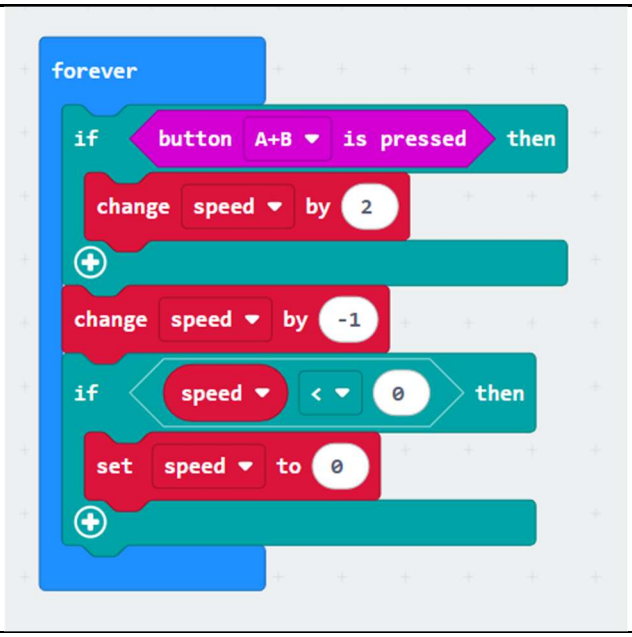
We now need to add another conditional statement to prevent our plane from having negative speed, since drag is always acting on the plane and reducing its speed.

Drag another *"if/then" block* from the *Logic tab* and place it below the previous block, still inside the *forever loop*. Then place a hexagonal *comparison block* inside the "if" block and select the *less-than* symbol.

Add the "speed" variable from the *red Variables tab* to the left side of the comparison, and type 0 on the right side. Your comparison should read: "If speed < 0".

Finally, place a *"set variable to"* block inside this "if" block, select *speed*, and set the value to 0.

Your code should now look like this:



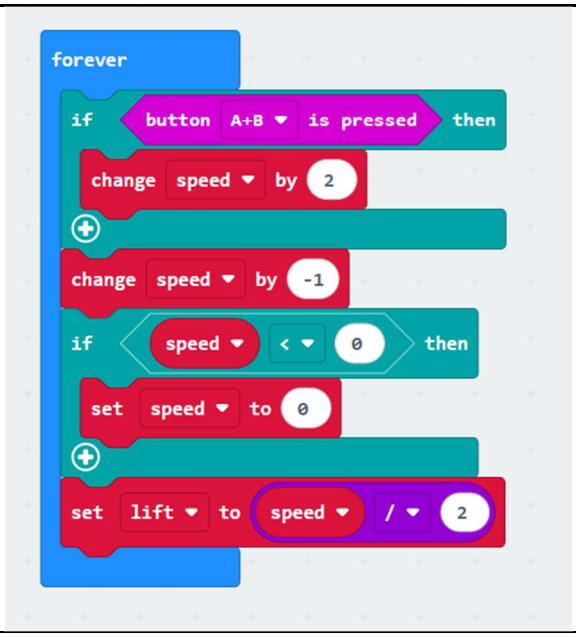
This means that our speed cannot go below zero, which would have happened otherwise since we just coded our drag force to always act on the plane.

3C: Adding lift.

The lift force is directly affected by the speed of our plane, so we need to code this relationship. To do this, place another "set variable to" block from the *red Variables tab* under the previous "if" block, still inside the *forever loop*. Be sure to select the *"lift"* variable.

Next, go to the *purple Math tab* and drag in an oval *division block*. Place the "*speed*" variable (found in the *red Variables tab*) on the left side of the equation, and type *2* on the right side. Your equation should read "speed divided by 2".

Now your code should look like this:



This means that if we increase the speed by pressing the A+B buttons (representing thrust) our lift will increase by half the amount the speed is increasing.

3D: Comparing lift and weight

The next step is to code what tells us whether our plane is *flying up*, *flying down*, or *staying level*. *Lift* causes the plane to rise, while the force of *weight* (gravity) causes it to descend.

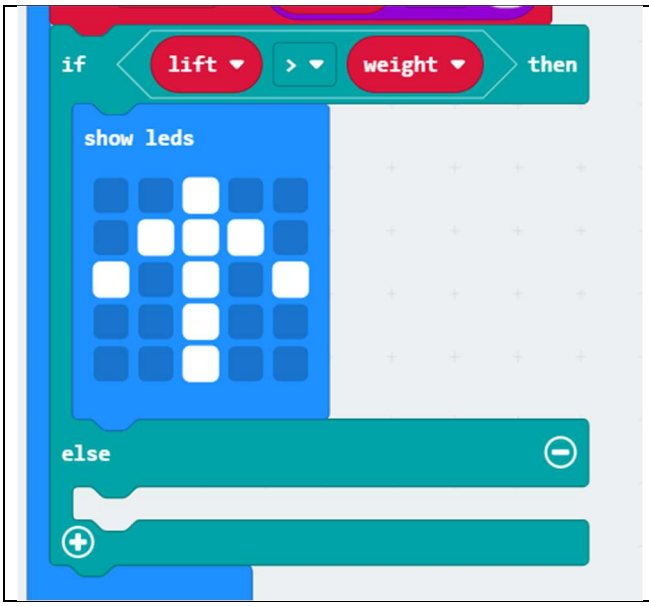
If the *lift force* is greater than the *weight*, the plane should go *up*.
If the *lift force* is less than the *weight*, the plane should go *down*.
If the *lift force* equals the *weight*, the plane should stay *level*.

You can probably guess that we need more *conditional statements*. This time, grab an *"if/else" block* from the *teal Logic tab* and place it inside the *forever loop*, just below the last block you added.

Next, drag in a hexagonal comparison block from the *teal Logic tab*, select the *greater-than* symbol, and place it inside the "if" portion of the block. On the left side of the comparison, add the *"lift" variable*; on the right side, add the *"weight" variable* (both from the *red Variables tab*). Your condition should read: "If lift is greater than weight".

Since a lift force greater than weight means the plane should go up, we need a way to show this on the Micro:bit. Add a *"show LEDs"* block from the *blue Basic* tab and design an up-arrow icon by selecting the LEDs you want to light up.

The new portion of your code should look like this:



Now, if the lift is greater than the weight, we will see an up-arrow displayed, indicating that our plane is gaining elevation.

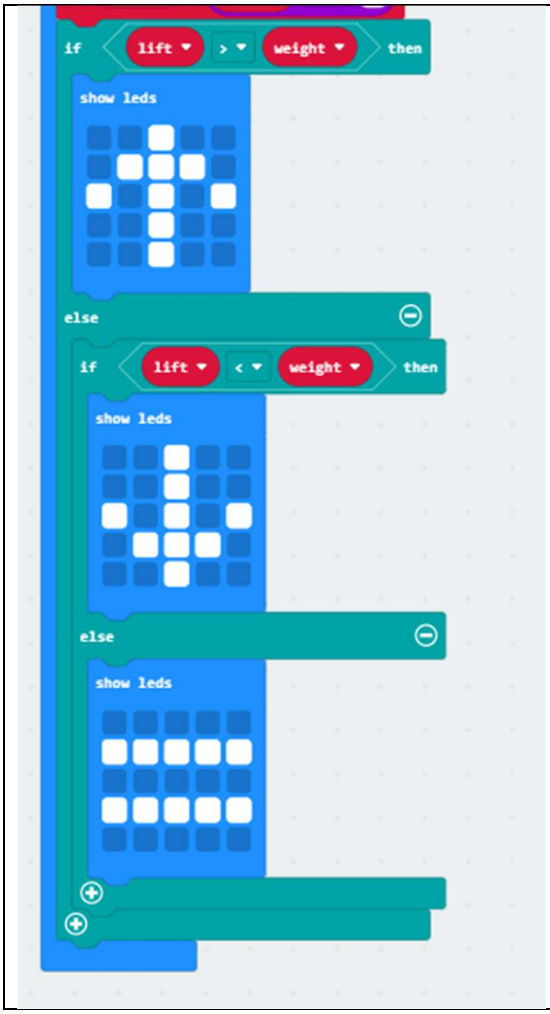
Now we need to complete the conditional statement. If the lift is not greater than the weight, there are two possibilities: it may be less than the weight, or it may be equal to the weight. We need to code for both scenarios.

To do this, place another *"if/else" block* from the *teal Logic tab* inside the *else* section of the first block.

Next, drag in another hexagonal *comparison block* from the *teal Logic tab* and select the *less-than* symbol. Place the *"lift"* variable on the left side of the comparison and the *"weight"* variable on the right. Your condition should read: "if lift is less than weight".

For this condition, follow the same steps as before—add a *"show LEDs"* block and design a *down-arrow* icon to indicate that the plane is descending.

The *else* portion of this second block represents the final possibility: lift equals weight. So we will drag another *"show LEDs"* block into the *else* section and select the LEDs to form an *equals sign*, indicating level flight.



This means that if the force of our lift is greater than that of the weight, our simulator will indicate our plane is gaining elevation with an "up arrow".

If the force of our lift is less than that of the weight, our simulator will indicate our plane is losing elevation with a "down arrow".

Else, the force of the lift must equal that of the weight, and our plane will neither gain nor lose elevation, it will fly level, indicated to us by an "equal sign".

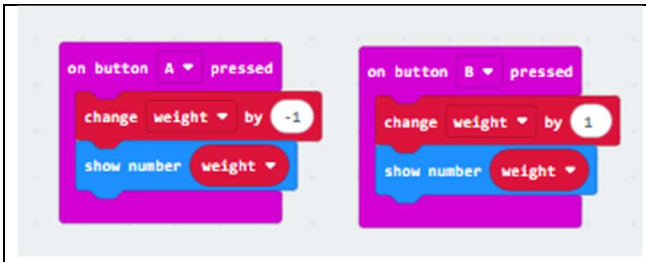
Step 4: **Controlling the weight**

Drag an *"on button pressed"* block from the *pink Input tab* and select "Button A". Inside this block, place a *"change variable by"* block from the *red Variables tab*, and select the variable *"weight"*. Change the value to –1.

Below that, still inside the same "on button pressed" block, add a *"show number"* block from the *blue Basic tab*. Drag the oval *"weight"* variable into the "show number" block to complete the code.

Repeat the same steps, except this time select *Button B* and change the value of the *"weight"* variable by 1 instead of –1.

Your additional code should look like this:

	This means that when we *press Button A*, the *weight* increases by *1* and the updated value is displayed on the screen, and when we *press Button B*, the *weight* decreases by *1* and the value is displayed on the screen.

Step 5: Optional codes

Each of the following code additions can enhance the simulation, but they are not required for the basic functionality. They can be included to add clarity or depth or omitted in the interest of simplicity and/or time.

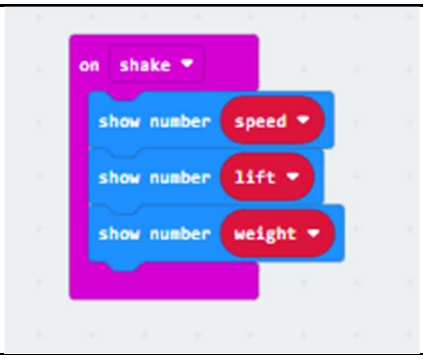
5A: Variable Value Indicator

It can be helpful to have a quick way to see the current values of your variables at any point during the simulation.

Drag an *"on shake"* block from the *pink Input tab*. Inside this block, place three *"show number"* blocks from the *blue Basic tab*, each containing one of the variables.

Be sure to remember the order in which you place the variables, this will help you interpret the values correctly when the Micro:bit displays them.

The code should look like this:

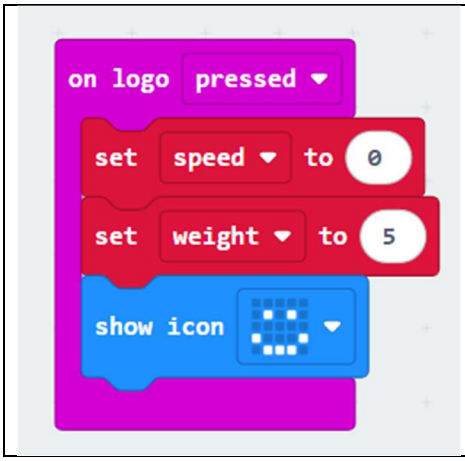
	This means that when we shake the microbit, the value of the speed, lift and weight will be shown sequentially on screen.

5B: Reset button

If you have the *version* 2 Micro:bits or are only using the online software, you also have a *Logo* that can be coded like a button. We can use this feature as a reset button to return our variables to their starting values.

Add an *"on logo pressed"* block from the *pink Input tab*. Inside it, place two *"set variable to"* blocks from the *red Variables tab* and select the variables *"speed"* and *"weight"*. Set *speed* to *0* and *weight* to *5*. Then add a *"show icon"* block from the *blue Basic tab* and choose any icon you'd like to represent the reset action.

Your code should look like this:

	This means that when you press the logo, your variables will reset to what we started with and show an icon of your choosing to let you know they have been reset.

Step 6: Experiment!

Download the code onto your micro:bits or use the provided simulator on the website next to your coding workspace. Let the students experiment with pressing the thrust (A+B buttons) and seeing how long it takes to get an up arrow, indicating the plane will be gaining elevation. Have them experiment with holding

the buttons, letting them go, etc. See what happens when you increase the weight (button B) and decrease the weight of the plane (button A).
Consolidation/Extension You can have the students create a chart to compare how many seconds it takes of adding thrust for a given weight to get the plane to “take off” and increase in elevation.