Unit 3: BRIGHTNESS, IF, and WHILE

Skill Builder 1: BRIGHTNESS measurements

In this lesson, we investigate the onboard light sensor, BRIGHTNESS, and use the **Disp** statement to show the sensor's readings.

Objectives:

- Read the BRIGHTNESS sensor
- Introduce the **While** loop
- Introduce **string**( ) and concatenation

In the previous lessons, we have only been sending instructions to the TI-Innovator™ Hub to have an impact on it's built-in devices (LIGHT, COLOR, and SOUND).

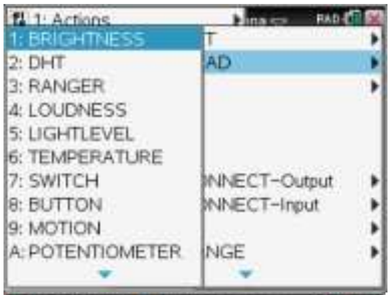
In this unit, we will work with the onboard light sensor, and use the value in our program to create a 'light meter'. The light sensor produces values in the range 0 to 100 in decimal form.

Obtaining the light level value from the TI-Innovator Hub requires TWO statements:

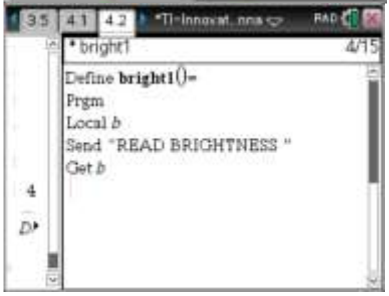
- **Send "READ BRIGHTNESS"**
- **Get** <*var*>

Setting up the program:

1. Start a new program. We call it BRIGHT1.
2. We'll use the variable ***b*** to store the BRIGHTNESS value, so we declare this variable to be **Local** to the program. This is optional.
3. Select **menu > Hub > Send "READ… > BRIGHTNESS**. Press enter.



4. Select **menu >** ***Hub*** **> Get**.
5. Type the variable name ***b*** (no parentheses are used here).



```
* bright1
Define bright1()=
Prgm
Local b
Send "READ BRIGHTNESS "
Get b
```

How it works:

- **READ BRIGHTNESS** tells the TI-Innovator Hub to read the brightness level and store that value in an onboard 'buffer'.
- **Get b** is a command to get the value from TI-Innovator Hub's buffer. This statement transfers the value from the buffer on the TI-Innovator Hub to the variable ***b*** in the TI-Nspire. The variable you use can be any legal variable name.

> **Teacher Tip:** A 'buffer' is a memory slot on the TI-Innovator Hub that temporarily stores a value. It is updated whenever another *READ* command is processed, so it is highly recommended to *READ* and immediately *Get* the value from the TI-Innovator Hub into a variable on the calculator. It is possible to collect a set of data from the TI-Innovator Hub and store that data into a list for future data analysis, but that is beyond the scope of this introduction.

While Loop:

The **While…EndWhile** loop (**menu > Control >**) is used to process a block of code while a *condition* is true. A *condition* is a logical statement that can be evaluated as true or false. The relational operators and the logical operators are found on the ctrl- = key.

The relational operators are **=**, **≠** **<**, **>**, **≤**, and **≥**.

The logical operators are **and**, **or**, **not**, and **xor**.

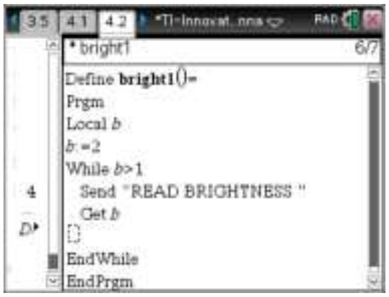
These operators can be used together to build compound conditions such as **x>0** and **y>0.**

We are going to use a simple **While** loop that stops when the BRIGHTNESS value is less than 1. To terminate the program, simply cover the light sensor on the end of the TI-Innovator Hub with your hand.

Adding a While Loop:

6. Before the **Send** statement in your program, add the statements:
    - **b:=2**
    - **While b>1** [use ctrl-= key to select **>**]

   These statements initialize the loop. As long as the condition **b>1** is true, the loop continues reading the light sensor. Once it becomes false, such as when there is no more light coming into the sensor with a hand covering it, the loop and the program terminate.



```
Define bright1()=
Prgm
Local b
b:=2
While b>1
  Send "READ BRIGHTNESS "
  Get b

EndWhile
EndPrgm
```

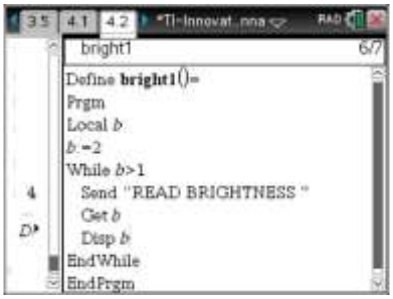
If you *select* **While…EndWhile** *from the Control menu, then the* **EndWhile** *statement will be in the wrong spot in your code!*

It belongs <u>*after*</u> *the* **Get** *statement as shown above. You can select it, cut it (ctrl-X), and paste it (ctrl-V) into the right place, or just delete it and type it into the correct place.*

The **EndWhile** of the **While** loop must also be entered. Below the **Get** statement, add the **EndWhile** statement for the end of the **While** loop.

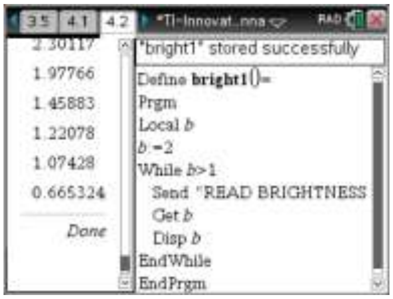
> **Teacher Tip:** It's always a good idea to give your programs a way to get out of a loop. Recall that each control structure has its own ***End…***. In larger programs, there will be many instances of these **End…** statements. The program processor in the calculator 'knows' which **End**… belongs with which control structure, but it is up to the programmer to design the correct code.

6. Add the **Disp** statement *after* the **Get** statement and *before* the **EndWhile** of the loop as shown by selecting **menu** > **I/O > Disp**.
7. Display the value of the variable ***b***.



```
Define bright1()=
Prgm
Local b
b:=2
While b>1
  Send "READ BRIGHTNESS "
  Get b
  Disp b
EndWhile
EndPrgm
```

8. Store the program (**ctrl-B**), and run the program in the Calculator app with the TI-Innovator Hub attached.
    - You should see a sequence of values scrolling down the Calculator app. These values change depending on the light intensity being read by the sensor.
9. To terminate the loop (and the program), cover the light sensor on the end of the TI-Innovator Hub so that the brightness value displayed is less than 1.



```
2.30117
1.97766
1.45883
1.22078
1.07428
0.665324
Done
```

```
"bright1" stored successfully
Define bright1()=
Prgm
Local b
b:=2
While b>1
  Send "READ BRIGHTNESS
  Get b
  Disp b
EndWhile
EndPrgm
```

Teacher Tip: Local variables exist only for the program run. They are not created in the current problem. If a variable used in the program is not declared local then, when the program runs, that variable will be created in the current problem. Sometimes this is an advantage, and sometimes it can be problematic.