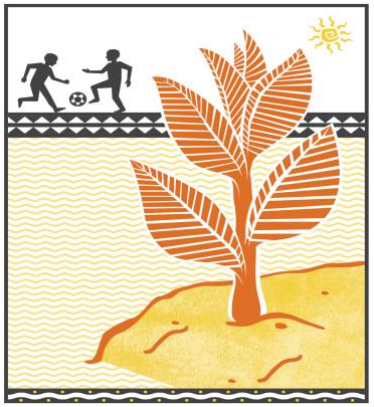
Smart Irrigation System



In this TI-Innovator™ project, you will design a smart irrigation system that could be used to monitor and meter water from a rain collection cistern that might be used to irrigate a small family garden in Zimbabwe. This model also applies to other related scenarios where “smarter water” usage makes sense. For example, water restrictions are often put in place during the hot summer months in areas of excessively hot and dry climates. A smart water irrigation system could alleviate some of these restrictions as well, if people are smarter about the way they water their yards.

You will have to utilize math skills, computer programming and engineering to design and build a smart watering system to solve a real-world problem like the drought in Zimbabwe or, even the problem right in your backyard!

Background:

Drought in southern Africa has caused a famine in Zimbabwe, and local students are quitting school and soccer to stay home and help grow food for the family. What can be done differently regarding the watering of crops to allow students to stay in school instead of working the fields?

Your Challenge:

You are challenged to help solve this problem by designing and building a smart irrigation system to manage a limited amount of water collected in a cistern to irrigate a garden.

Activity Materials:

- TI-Nspire CX family calculator
- TI-Innovator Hub
- Grove - Temperature & Humidity Sensor
- Grove – Soil Moisture Sensor
- Grove – Light Sensor
- Grove – MOSFET for power control module with 4xAA battery holder
- 4 x AA batteries to provide power to the pump (required)
- Water Pump with the plastic tube
- Possible supplies for building garden model:
- Drinking straws
- Duct Tape (always useful)
- Container for the plants, such as a 1-gallon milk
- Soil, perlite or some other growth medium
- Fast germinating seeds like radish, lettuce or similar.

Project Challenges

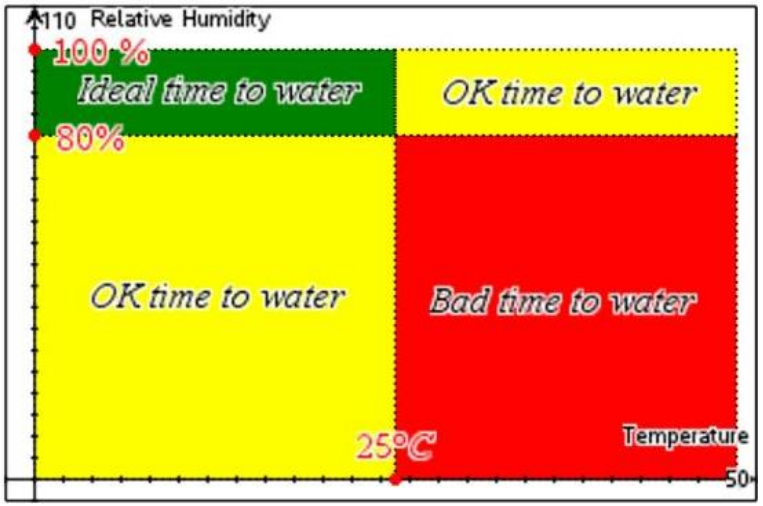
Skill Building Challenge 1: Write a program named *c1* that continuously measures and displays the ambient light level. The program should:

1. connect a light level sensor to IN 1.
2. range the light level reading from 0 to 100.
3. use a while loop to continuously read and display light level every ½ second on the calculator screen.
4. enable the ESC key to quit the while loop and end the program.

Skill Building Challenge 2: Write a program named *c2* that measures soil moisture every two seconds for a total of twenty times. The program should display if the soil is dry or moist based on the sensor reading.

Skill Building Challenge 3: Write a program named *c3* that connects the Digital Temperature and Humidity (DHT) sensor. Read 20 measurements at two-second intervals and display with an appropriate message.

1. Use a decision tree based on the temperature and relative humidity measurements to determine the present watering conditions as indicated in the graph below. Display an appropriate message with each of the four cases.



Skill Building Challenge 4: Write a program named *c4* using a While loop to continuously measure and display a dashboard of all of sensor value readings. The user should be able to stop the monitoring by pressing the ESC key.

Skill Building Challenge 5: Write a program named *c5* to connect the pump power module and run the pump for 20 seconds. Be sure to turn the pump off.

1. Try setting the pump power to different values.
2. Try to estimate the pump flow rate in mL/sec.

Final Challenge: Write a program that uses all of the sensors in the previous challenges. Continuously monitor light level, soil moisture, temperature, and humidity and display the current value on the display. Use your knowledge of ecology, biology, and earth science to determine what the best conditions are to water your garden. When the condition is correct, set the pump to deliver water at a rate that is best for your garden.

Test your system with these conditions:

- light level < 20
- soil moisture < 10
- temperature < 25
- humidity > 80

When these conditions are true, set the pump on to 255. When the conditions are false, set the pump to zero.

Example Programming Statements for the project

Example Code	Behavior
`from ti_hub import *`	Imports all the functions in the ti_hub module for use in the program. The ti_hub module includes all the necessary additions needed for the Mood Ring project.
`# defines a light_level sensor` `# object named lightsensor`	# at the beginning of a line denotes a comment. Comments are a "best practice" by programmers to annotate their code. Comment statements are ignored when the program is run. In the TI-Nspire CXII Python editor, [ctrl]+[T] toggles the statement of the current cursor location from a comment to a statement that will be run.
`lightsensor=light_level("IN 1")`	Creates a light_level sensor object named **lightsensor** connected to port IN 1. light_level is available from the TI Hub > Add Input Device menu. Note: = is the Python operator for storing or assigning values to a variable.
`lightsensor.range(0,100)`	Scales the measured values read from the **light.sensor** object to the range of 0 to 100. **Note:** to see options for an object select the object name from the var key menu then press the "." key.
`light=lightsensor.measurement()`	Reads and stores the current measurement value of the **lightsensor** object into variable **light**. Note: **.measurement()** returns the current measured value of a sensor object. To see options for an object select the object name from the var key menu then press the "." key.
`text_at(3,"light level= " +str(light),"left")`	This text_at() function displays a text string on a specified row with an alignment of left, center or right. When variable **light** has a value of 26, the following is displayed on row 3, aligned to the left: light level = 26 text_at() is available from the TI Hub>Commands menu. **Note:** The **str()** function converts a numeric value to a string. The **+** operator is used to join two strings. **str()** is available from the Built-ins> Type menu. **Note:** Degree, percent and other special characters are available from the ?! key menu in the lower right of the TI-Nspire keyboard.

`while get_key() != "esc":`
`  light=lightsensor.measurement()`
`  text_at(3,"light level= "+str(t),"left")`
`  sleep(1)`
	Defines a while loop that will continue until the escape key is pressed.
	While loops repeat the statements in the block if the condition at the top of the loop is true. In the example, looping continues until the escape key is pressed. Not pressing a key or pressing any key but escape means that get_key() will return a value that is not equal to "esc". The loop condition is true and looping continues. If the escape key is pressed, get_key() returns "esc". The condition will evaluate as "esc" not equal to "esc", which is false. A false result means that the loop statements are not repeated. Program execution skips to the statement just after the loop. Note: The block starts with a **colon** and includes the indented lines that follow. while get_key() != "esc": is available from the TI Hub > Commands menu.

`sleep(.5)`	Pauses program for .5 seconds.
`for n in range(10):` `  print(n)`	Repeats the statements in the block ten times, printing the value of the index variable, **n**, as 0,1,2,…9. The index variable n starts at 0 and increases by 1 with each loop. If **n** is less than the stop value, 10, the loop continues to repeat.

`2+3==6 (result is false)`
`x+4>=y (if x=1 and y=3, the result is true)`
`"enter"!="esc" (result is true)`
	Boolean expressions evaluate to either true or false. The examples show some of the relational operators available from the Built-ins > Ops menu.
	Note: == is the Python operator to check equality. >= is the Python operator to check whether the value to the left is greater than or equal to the value on the right. != is the Python operator to check inequality.

`if moisture < 10:`
`  text_at(5,"The soil is dry","left")`
`else:`
`  text_at(5,"The soil is moist","left")`
	Checks to determine if the value of variable **moisture** is less than 10. If the statement is "true" then the statements in the **if** block are executed. If the statement is "false" then the statements in the **else** block are executed. In the example, when the **moisture** value is less than 10, the text "The soil is dry" will be displayed on the calculator screen. When the value of moisture is 10 or greater the text "The soil is moist" will be displayed. Note: **if..else..** is available from the Built-Ins > Control menu.

`if temperature<= 25 and humidity>=80:` `  text_at("it is cool and humid.")`	If both expressions are true the **and** function is "true", then the block is executed. Otherwise, the **and** function returns false, and the block is skipped. In the example, when the temperature is less than or equal to 25 and the humidity is greater than or equal to 80, the message "It is cool and humid" will be printed on the calculator screen.
`pump=analog_out("OUT 1")`	Creates an analog_out object named **pump** connected to port OUT 1. analog_out is available from the TI Hub > Add Output Device menu. Note: = is the Python operator for storing or assigning values to a variable.

`pump.set(255)`
`sleep(10)`
`pump.set(0)`
	Sets the value of an analog_out device named **pump** to 255 (full power). The pump continues at full power until a value of 0 (off) is received after the sleep function causes the program to pause for 10 seconds. **Note:** The output device will continue using a setting value until a new setting value is received or until the power to the system is removed. Make sure to use the .set(0) or .off() output objective methods to turn the device off. To see options for an object select the object name from the var key menu then press the "." key.

`speed=int(input("pump speed (0-255)= "))`	Prompts the user to enter a value that will be stored as an integer to the variable **speed**. input() prompts the user to enter a text string. Int() from the. Built-Ins>Type menu changes the text string to an integer data type. The integer value is stored to the variable **speed** for later use in functions that require numeric values as inputs.

Calculator Notes:

- On the Home screen press 4:Current (or the Home key) to return to your document file.
- On the Home screen Press 1:New to create a new document file.
- You create and edit programs in a Python app program editor page. You run programs from a Python app Shell page.
- Use the [menu] key to see the options for your current app.
- ctrl-B is the shortcut from the Run menu to check the syntax and save changes to your program.
- In the Python editor ctrl-R is the shortcut from the Run menu to check the syntax and save changes and to run the program on the following Python Shell page.
- Use ctrl-R in the Shell to re-run a program.
- Press [enter] to run the statement entered on the line of a Python Shell page.
- Find your function names in the Python app by pressing the [var] (variables) key.
- To see options for an object paste the object name from the var key menu into the Python editor then press the period key.
- Move from page to page by using ctrl-left arrow and ctrl-right arrow or by using the touchpad pointer to click on the desired page tab.
- ctrl-doc (+page) will add a blank page to your document.
- ctrl-Z will undo your last action.
- To stop (“break”) a program press and hold the ON key until you receive a dialogue box.
- ctrl-S is the shortcut for saving your entire document file. Do this periodically to save your work.

Sensor and actuator Hub connections