



#### Unit 2: for loops with the TI-Innovator™ Hub

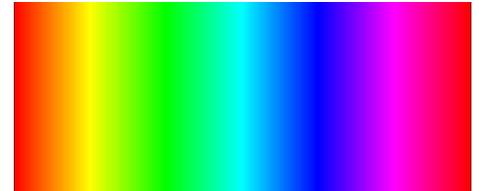
#### Skill Builder 2: Loop through Color

In this lesson, you will learn about color mixing to make a variety of colors on the color LED using several **for** loops.

#### Objectives:

- Use the **for** loop with the **range()** function
- Control the many colors possible on the color LED
- Use copy/paste/edit for simpler coding of longer programs

All the colors of the rainbow (and more) are possible using the color LED on the TI-Innovator Hub. This lesson builds a program that will change the color LED to many different colors using **for** loops. You will get some practice with the different loop options.



1. Start a new Python Hub Project.

This program will gradually increase and decrease the LED's color channels, one at a time, to mix some colors.

Write two **input** statements to enter a **step** value and a **delay** value. The **step** value can be an **int()** but the **delay** should be a **float()** since it may need to be between 0 and 1 (a decimal value).

**step** is the space between color values and **delay** is the time delay for each color step.

```

1.3 1.4 1.5 *Unit 2 Py...ops RAD 11/36
*u2sb2.py
#-----
from ti_hub import *
from math import *
from time import sleep
from ti_plottlib import text_at,cls
from ti_system import get_key
#-----
step=int(input("Step? "))
delay=float(input("Delay? "))

```

2. Press **menu > Built-ins > Control** and notice the **for index in...** section of the menu (shown at the right).

All these statements use the **range()** function but have different arguments. The most versatile of the three is:

```

4 for index in range(size):
5 for index in range(start, stop):
6 for index in range(start, stop, step):

```

### **for index in range(start, stop, step):**

**Teacher Tip:** **for** loops that use **range()** are counting loops. The **step** can be either a positive or negative number. The **stop** value is not processed in the loop, but must eventually be reached or the loop will never terminate (example: `for i in range(10,5,2)` will produce 10, 12, 14, 16, 18, ...).

3. Select the statement

### **for index in range(start, stop, step):**

and change index to the variable **i**. Enter 0 for **start**, 256 for **stop**, and **step** for **step**. (You must type the word 'step' since this is the variable used in the input statement.)

This makes the loop go from 0 up to at most 255 by adding **step** in each step of the loop. For example, if **step** is 10 the values of **i** will be

0, 10, 20, 30, ... up to?

```

1.2 1.3 1.4 *Unit 2 Py...ops RAD 12/57
*u2sb2.py
from ti_hub import *
from math import *
from time import sleep
from ti_plottlib import text_at,cls
from ti_system import get_key
#-----
step=int(input("Step? "))
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
=>=block

```



# 10 Minutes of Code - Python

TI-NSPIRE™ CX II WITH THE TI-INNOVATOR™ HUB

## UNIT 2: SKILL BUILDER 2

### TEACHER NOTES

4. For the loop **block** use

**color.rgb(i, 0, 0)** (red only)

found on **menu > TI Hub > Hub Built in Devices > Color Output**.

Press **ctrl+R** to run the program. Notice that the LED lights up immediately. It is supposed to light up *slowly*. You need to slow the process down a bit using your **delay** variable. Try it yourself before going to the next step.

```

1.2 1.3 1.4 *Unit2 Py_ops RAD 12/62
* u2sb2.py
from math import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
#=====
step=int(input("Step? "))
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
    color.rgb(i,0,0)

```

5. Did you add a **sleep(delay)** statement?

Now the LED *gradually* adds red. Next write another **for** loop to *gradually* add green to the red LED.

You can *copy* the red **for** loop and *paste* it below the loop and then *edit* the text to control only the green channel.

Be careful about the indenting.

Make sure the red channel stays *fully* lit the whole time.

```

1.2 1.3 1.4 *Unit2 Py_ops RAD 6/62
* u2sb2.py
from ti_plotlib import text_at,cls
from ti_system import get_key
#=====
step=int(input("Step? "))
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
    color.rgb(i,0,0)
    sleep(delay)

```

**Teacher Tip:** it's okay to re-use the loop variable *i*. Copy and paste? Just like on a computer:  
**Shift+arrows** to select text,  
**Ctrl+c** to copy, and then  
**Ctrl+v** to paste.

6. Here's the green loop:

**for i in range(0, 256, step):**  
**color.rgb(255, i, 0)**  
**sleep(delay)**

Note that the red channel is set to 255 and the loop variable is now in the green position.

Run the program again. What color do you see at the end of the program now?

```

1.2 1.3 1.4 *Unit2 Py_ops RAD 14/52
* u2sb2.py
from ti_system import get_key
#=====
step=int(input("Step? "))
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
    color.rgb(i,0,0)
    sleep(delay)
for i in range(0,256,step):
    color.rgb(255,i,0)
    sleep(delay)

```



7. Now for the tricky part...gradually *remove* the redness by changing the 'direction' of the **for** loop:

**for i in range(255, 0, -step):**

This loop starts at  $i=255$  and the loop index  $i$  decreases by the step value.

```
*u2sb2.py 22/53
for i in range(0,256,step):
    color.rgb(i,0,0)
    sleep(delay)
for i in range(0,256,step):
    color.rgb(255,i,0)
    sleep(delay)
for i in range(255,0,-step):
    color.rgb(i,255,0)
    sleep(delay)

```

**Teacher Tip:** It gets close to 0 but will not use the value 0 in the loop. Depending on the step, it will stop just *before* it reaches 0. Remember that the loop ends when the index *equals* or passes the stop value.

8. It might help to keep things organized by adding **#** comments before each **for** statement to explain what it does more clearly. Comments begin with the **#** symbol (press **ctrl+T**) and are ignored when the program is run.

To complete this project, add loops to:

- Increase blue
- Decrease green but keep blue
- Increase red (again)
- Decrease blue but keep red
- Decrease red to turn it off slowly

This process mixes all three pairs of colors: red-green, green-blue, and blue-red. In the process you should also see yellow, cyan, and magenta appear.

At the end of the program, the LED should be off.

Remember to save your work.

```
*u2sb2.py 19/55
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
    color.rgb(i,0,0)
    sleep(delay)
#increase green
for i in range(0,256,step):
    color.rgb(255,i,0)
    sleep(delay)
#decrease red but keep green
for i in range(255,0,-step):

```

**Teacher Tip:** The final code is rather long but entering it is facilitated by copy/paste/edit:

```
step=int(input("Step? "))
delay=float(input("Delay? "))
# increase red
for i in range(0,256,step):
    color.rgb(i,0,0)
    sleep(delay)
# increase green
for i in range(0,256,step):
    color.rgb(255,i,0)

```



```
    sleep(delay)
# decrease red
for i in range(255,0,-step):
    color.rgb(i,255,0)
    sleep(delay)
# increase blue
for i in range(0,256,step):
    color.rgb(0,255,i)
    sleep(delay)
# decrease green
for i in range(255,0,-step):
    color.rgb(0,i,255)
    sleep(delay)
# increase red
for i in range(0,256,step):
    color.rgb(i,0,255)
    sleep(delay)
# decrease blue
for i in range(255,0,-step):
    color.rgb(255,0,i)
    sleep(delay)
# decrease red then all off
for i in range(255,0,-step):
    color.rgb(i,0,0)
    sleep(delay)
color.off()
```