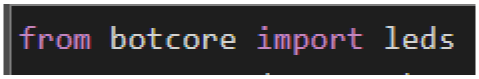
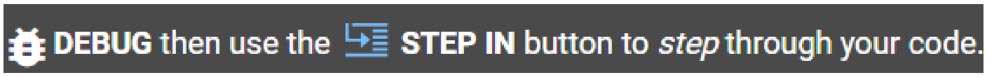
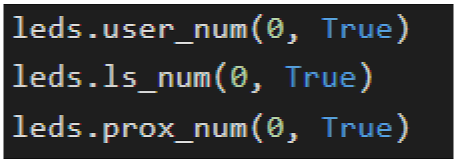
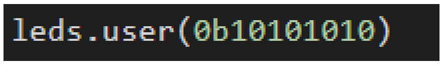
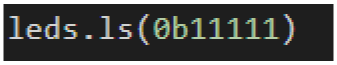
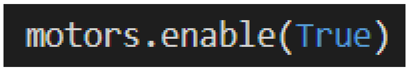
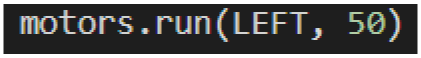
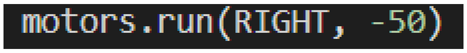
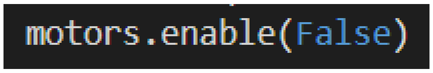
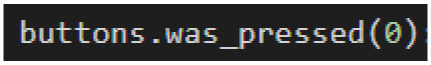
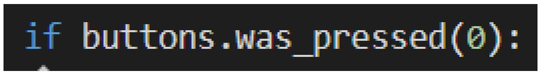
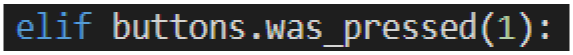
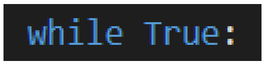
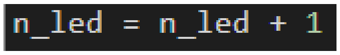
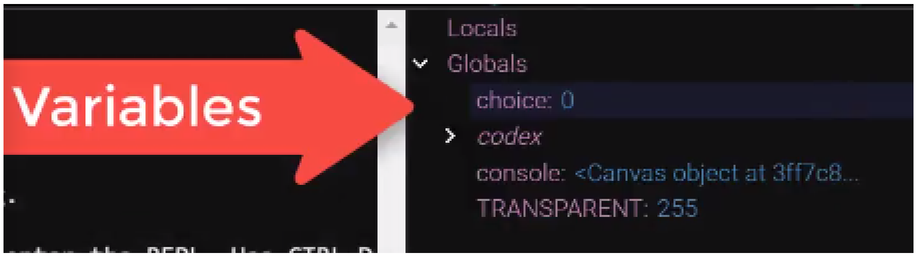
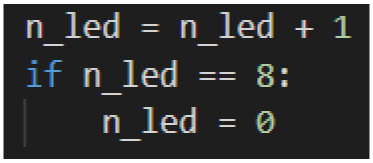
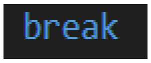
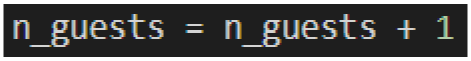
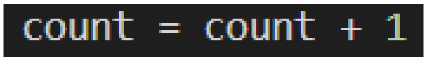
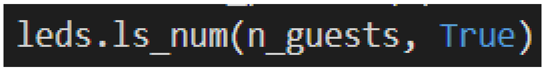
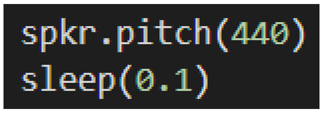
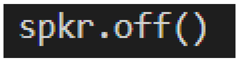
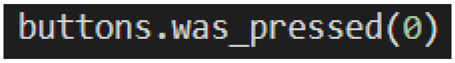
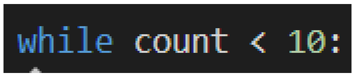
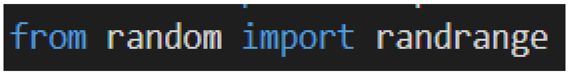
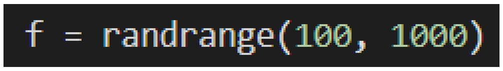
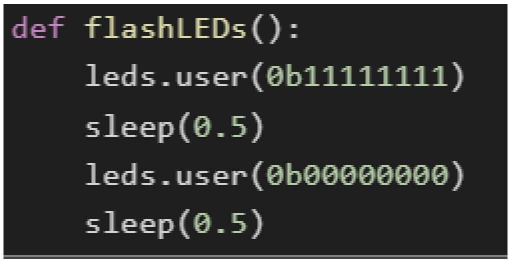
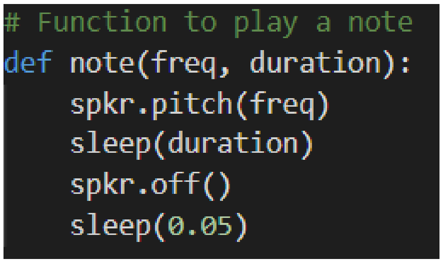
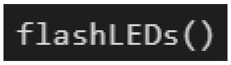
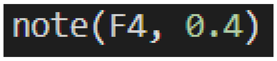
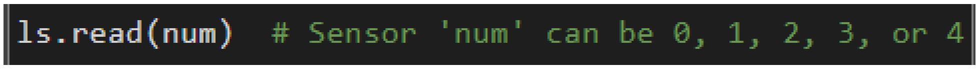
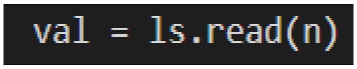
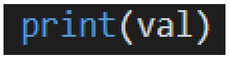
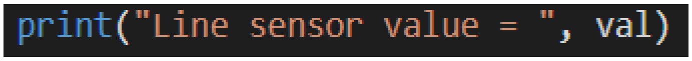
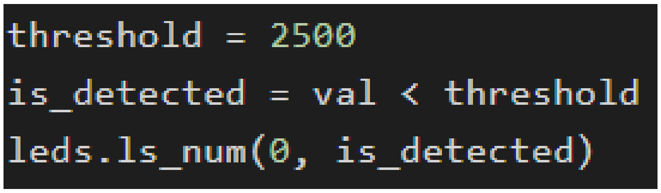
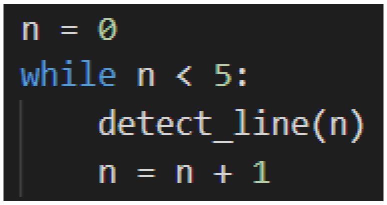
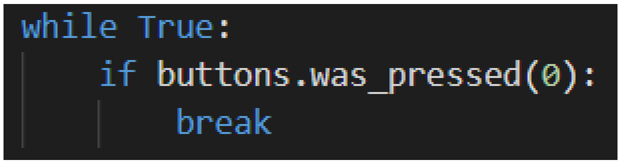
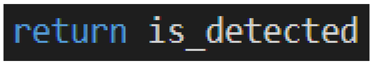
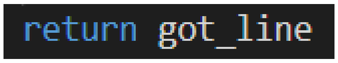
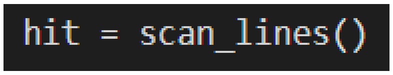
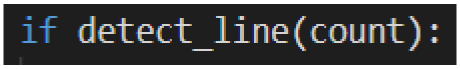
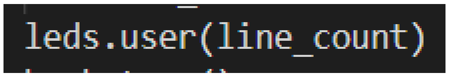
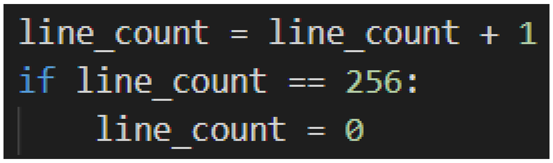
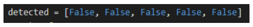
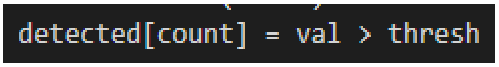
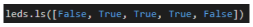
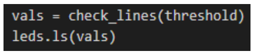
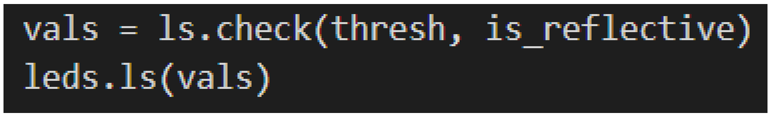
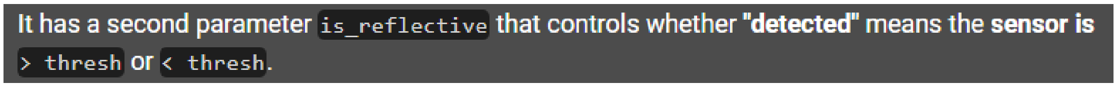
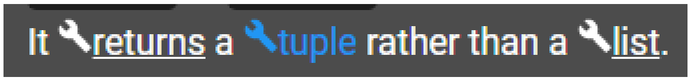
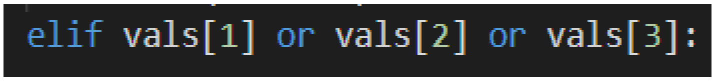
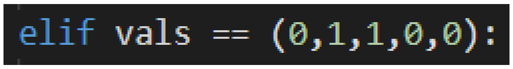
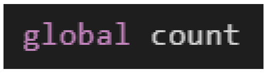
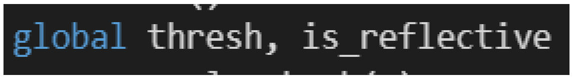
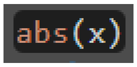
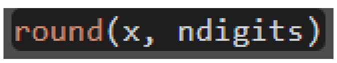
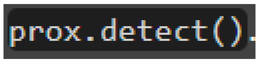
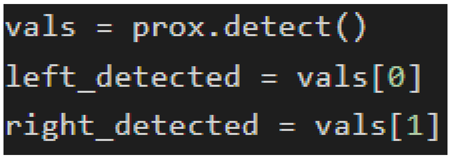
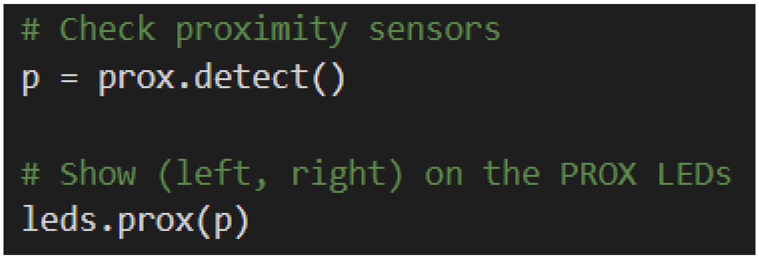
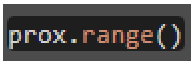
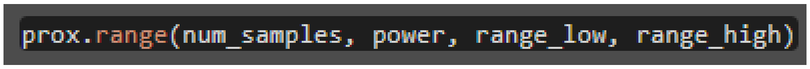
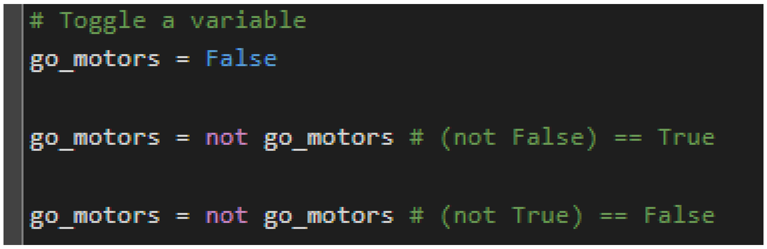
# Unit 1 CodeBot Python Code By Mission

| | |
|---|---|
| **Mission 2 – Introducing CodeBot** | |
| Import from botcore only leds functions |  `from botcore import leds` |
| Turn on one user LED | `leds.user_num(0, True)` – parameters are (LED number 0-7, True=on or False=off) |
| Line sensor LED | `leds.ls_num(0, True)` – parameters are (LED number 0-4, True=on or False=off) |
| **Mission 3 – Time and Motion (Objectives 1-6)** | |
| CodeSpace Debugger |  **DEBUG** then use the **STEP IN** button to *step* through your code. |
| Import a delay | `from time import sleep` |
| Use sleep() | `sleep(1.0)` – will sleep (amount of time in seconds) |
| Define a variable | `delay = 1.0` (define variables at the top of the code, just under import statements) |
| Use a variable with sleep() | `sleep(delay)` |
| Turn off an LED | `leds.user_num(2, False)` |
| Turn on three types of LEDs |  `leds.user_num(0, True)`<br>`leds.ls_num(0, True)`<br>`leds.prox_num(0, True)`<br>User LEDs (middle of the bot)<br>Line sensor LEDs (across the front)<br>Proximity sensor LEDs (one on each side) |
| Use binary designation for turning on LEDs |  `leds.user(0b10101010)` - 0b for binary, then 0=off, 1=on for each LED<br> `leds.ls(0b11111)` |
| **Mission 3 – Time and Motion (Objectives 7-9)** | |
| Import entire library | `from botcore import *` – * is a wildcard, which means everything |
| Turn on motors |  `motors.enable(True)` – must be done before motors will turn and wheels move |

| | |
|---|---|
| Power a motor |  `motors.run(LEFT, 50)` – will turn left wheel forward at 50% power<br> `motors.run(RIGHT, -50)` – will turn right wheel backward at 50% power |
| Turn off motors |  `motors.enable(False)` |
| **Mission 3 – Time and Motion (Objectives 10-11)** | |
| Returns Boolean value button was pressed |  `buttons.was_pressed(0)` – checks button 0, returns True (pressed) or False (not pressed) |
| Use button press in branching |  `if buttons.was_pressed(0):`<br> `elif buttons.was_pressed(1):` |
| **Mission 4 – Animatronics (Objectives 1-5)** | |
| Infinite loop |  `while True:` |
| Updating a variable |  `n_led = n_led + 1` |
| Use debugger to view variables |  <br>Locals<br>Globals<br>choice: 0<br>*codex*<br>console: <Canvas object at 3ff7c8...<br>TRANSPARENT: 255<br>Open the console panel while debugging |
| Reset a variable to stay within a range |  `n_led = n_led + 1`<br>`if n_led == 8:`<br>`    n_led = 0` |
| Break out of a loop |  `break` |
| Increment |  `n_guests = n_guests + 1`  `count = count + 1` |
| Turn on LED using a variable |  `leds.ls_num(n_guests, True)` |
| **Mission 4 – Animatronics (Objectives 6-12)** | |
| Play a tone on the speaker |  `spkr.pitch(440)`<br>`sleep(0.1)`<br>the (argument) is the pitch frequency |

<table>
<tr><td>Turn off the speaker</td><td>
<pre>spkr.off()</pre></td></tr>
<tr><td>Debounce a button press</td><td>
<pre>buttons.was_pressed(0)</pre></td></tr>
<tr><td>While loop</td><td>
<pre>while count &lt; 10:</pre>
(will iterate, or repeat, 10 times if count starts at 0)</td></tr>
<tr><td>Import random library</td><td>
<pre>from random import randrange</pre></td></tr>
<tr><td>Get a random number within a range</td><td>
<pre> f = randrange(100, 1000)</pre></td></tr>
<tr><td>Define a function</td><td>
<pre>def flashLEDs():
    leds.user(0b11111111)
    sleep(0.5)
    leds.user(0b00000000)
    sleep(0.5)</pre>

<pre># Function to play a note
def note(freq, duration):
    spkr.pitch(freq)
    sleep(duration)
    spkr.off()
    sleep(0.05)</pre></td></tr>
<tr><td>Call a function</td><td>
<pre>flashLEDs()</pre>

<pre>note(F4, 0.4)</pre></td></tr>
<tr><td colspan="2"><b>Mission 5 - Fence Patrol</b></td></tr>
<tr><td>Read a line sensor</td><td>
<pre>ls.read(num)  # Sensor 'num' can be 0, 1, 2, 3, or 4</pre>

<pre> val = ls.read(n)</pre>
(returns a value between 0 and 4095)</td></tr>
<tr><td>Display the value of a variable in the console</td><td>
<pre>print(val)</pre>

<pre>print("Line sensor value = ", val)</pre></td></tr>
<tr><td>Assign a Boolean result of a comparison to a variable<br><br>Use the Boolean variable in code</td><td>
<pre>threshold = 2500
is_detected = val &lt; threshold
leds.ls_num(0, is_detected)</pre></td></tr>
<tr><td>Detection</td><td>Dark line on light surface – use val &gt; threshold<br>Light line on dark surface – use val &lt; threshold</td></tr>
<tr><td>Use a comparison with a while loop and use the control variable as an argument in a function call</td><td>
<pre>n = 0
while n &lt; 5:
    detect_line(n)
    n = n + 1</pre></td></tr>
</table>

| | |
|---|---|
| Wait loop<br>(safe driving) | <br>`while True:`<br>`    if buttons.was_pressed(0):`<br>`        break` |
| Return statement | <br>`return is_detected`<br><br>`return got_line` |
| Call to a function that has a return | <br>`hit = scan_lines()`<br><br>`if detect_line(count):` |
| Use a variable to turn on LEDs | <br>`leds.user(line_count)`<br>line_count will be from 0 to 255 |
| Wrap-around the line_count variable for binary numbers | <br>`line_count = line_count + 1`<br>`if line_count == 256:`<br>`    line_count = 0` |
| **Mission 6 - Line Follower** | |
| Create a list | <br>`detected = [False, False, False, False, False]` |
| Update a specific value in a list | <br>`detected[count] = val > thresh` |
| Use a list with LEDs | <br>`leds.ls([False, True, True, True, False])`<br><br>`vals = check_lines(threshold)`<br>`leds.ls(vals)` |
| Botcore line sensors function (similar to check_lines) but faster | <br>`vals = ls.check(thresh, is_reflective)`<br>`leds.ls(vals)`<br>ls.check() takes 2 parameters<br><br>It has a second parameter `is_reflective` that controls whether **"detected"** means the **sensor is** `> thresh` or `< thresh`.<br><br>It returns a tuple rather than a list. |
| Using or (logical operator) | <br>`elif vals[1] or vals[2] or vals[3]:`<br>can have two or more conditions; if any of the conditions are true, the statement will evaluate to true |
| Comparing with a tuple | <br>`elif vals == (0,1,1,0,0):` |
| Code needed to change a global variable inside a function | <br>`global count`<br><br>`global thresh, is_reflective` |
| Built-in math operations | <br>`abs(x)`<br><br>`round(x, ndigits)` |

| **Mission 7 - Hot Pursuit** | |
|---|---|
| Read the proximity sensors |  `prox.detect().` returns a tuple (left, right) with values True or False<br> `vals = prox.detect()`<br>`left_detected = vals[0]`<br>`right_detected = vals[1]`<br>Index values: 0 = left 1 = right |
| Proximity LEDs |  `# Check proximity sensors`<br>`p = prox.detect()`<br><br>`# Show (left, right) on the PROX LEDs`<br>`leds.prox(p)` |
| Use parameters | **P = prox.detect(power, threshold)**<br>Power is the "'bot flashlight" with settings from 1 to 8 (high power)<br>Threshold is the sensitivity level, with settings from 1 to 100 (how much light is needed to detect) |
| Another built-in function that finds the ideal thresh for a given environment |  `prox.range()`<br> `prox.range(num_samples, power, range_low, range_high)`<br>All parameters are optional |
| Toggle the motors on and off – can be used with a button press to turn on/off the motors |  `# Toggle a variable`<br>`go_motors = False`<br><br>`go_motors = not go_motors # (not False) == True`<br><br>`go_motors = not go_motors # (not True) == False` |
| | |
| | |