Unit 3 CodeBot Python Code By Mission

Mission 5 Lesson 1 - Fence Patrol	
`val = ls.read(0)`	Read a line sensor. The number of the line sensor is the (argument). It returns an integer between 0 and 4095.
`print(val)`	Print the value of a variable to the console panel.
`print("Line sensor value = ", val)`	Print the value of a variable with a text message.
Mission 5 Lesson 2 - Fence Patrol	
`is_detected = ls.read(0) > threshold`	Assign a Boolean value to a variable.

`while True:`
`  # Read line sensor 0`
`  is_detected = ls.read(0) > threshold`
`  leds.ls_num(0, is_detected)`
	A use of Boolean value for turning on/off an LED

Surface Detection	Dark line on light surface – use val > threshold Light line on dark surface – use val < threshold

`def detect_line(n):`
`  is_detected = ls.read(n) > threshold`
`  leds.ls_num(n, is_detected)`
	Define a function with a parameter for detecting a line.

`detect_line(0)`	Call the function for a single, specific line sensor

`n = 0`
`while n < 5:`
`  detect_line(n)`
`  n = n + 1`
	While loop that repeats 5 times.
	It uses n as the control variable, which is initialized outside the loop and incremented inside the loop. It is used to determine which line sensor to read and which LED to turn on/off.

`def scan_lines():`
`  n = 0`
`  while n < 5:`
`  detect_line(n)`
`  n = n + 1`
	A function that calls another function.

Mission 5 Lesson 3 - Fence Patrol	

`while True:`
`    if buttons.was_pressed(0):`
`        break`
 	
`motors.enable(True)`
	(Review) Wait loop.
	A safety feature; the 'bot waits until BTN-0 is pressed before continuing the program.

`return is detected` `return got_line`	Function return The value of the variable is returned to the function call.
`hit = scan_lines()` `if detect_line(n):`	Function call The value of the return is used in the assignment or if statement.
`leds.user(line_count)`	Use a variable to turn on user LEDs. Line count needs to have a value from 0 to 255.

`line_count = line_count + 1`
`if line_count == 256:`
`    line_count = 0`
	Reset a counter variable when it reaches its maximum number.

Mission 5 Lesson 4 - Fence Patrol	

`def go_forward():`
`    motors.run(LEFT, 45)`
`    motors.run(RIGHT, 45)`
	Define a function for movement.

`else:` `    go_forward()`	Call a function for movement.
Mission 6 Lesson 1 - Line Follower	
`a_list = [4, 2, 5, 3, 6, 9, 1, 0]` `detected = [False, False, False, False, False]`	Define a list
`num_items = len(a_list)`	Length of a list (number of items)
`first_item = a_list[0]`	Access a single item in a list
`if is_detected:` `    detected[n] = True`	Update the item at index n
`leds.ls([True, True, False, False, False])`	Use a list to turn on/off LEDs

`vals = check_lines(2000)` `leds.ls(vals)`	Return a list of Boolean values, and then use the list to turn on/off LEDs.
Mission 6 Lesson 2 - Line Follower	
`vals = ls.check(2000, False)` `vals = ls.check(thresh, is_reflective)`	Botcore line sensor function (similar to check_lines but faster) Thresh is the threshold for detecting a line. False for a black line, True for a white line. The function returns a tuple of bools (not a list)
`ls.check(0)`	Returns the line sensor readings when entered in the Console Panel.
`elif vals[1] or vals[2] or vals[3]:`	Uses the logical operator “or” for multiple conditions. If any condition is true, the statement will evaluate to true.
Mission 6 Lesson 3 - Line Follower	

`def drive(left, right):`
`  motors.run(LEFT, left)`
`  motors.run(RIGHT, right)`
	Define a function for driving the ‘bot that uses parameters for the left and right wheel speeds.

`if vals == (1, 0, 0, 0, 0):` `  drive(0, 30)`	Use the tuple in a comparison. Call the drive function, selecting left and right speeds as the arguments.
`sensors = ls.check(0)`	Return a tuple of integers for the line sensor readings.
`if abs(gap) < 500:`	Use the absolute value function.
`is_reflective = line < ground`	Use a condition to set a Boolean value.
`thresh = round(ground + (gap/2))`	Use the round function; the result is an integer.
`global thresh, is_reflective`	Used at the beginning of function, the “global” keyword makes the variables global instead of local. The variables can be listed in any order.
`elif buttons.was_pressed(1):` `  calibrate()`	Call a function when a button is pressed.