Unit 1 CodeBot Python Code By Mission/Lesson

Mission 2 Lesson 2 – Introducing CodeBot	
`from botcore import leds`	Import from botcore only leds functions
`leds.user_num(0, True)` `leds.user_num(7, False)`	Turn on one user LED Parameters are (LED number 0-7, True=on or False=off)
`leds.ls_num(0, True)`	Turn on one line sensor LED Parameters are (LED number 0-4, True=on or False=off)
Mission 3 Lesson 1 – Time and Motion (Objectives 1-5)	
DEBUG then use the STEP IN button to *step* through	CodeSpace Debugger
`from time import sleep`	Import the ability to delay, or pause, the code
`sleep(1.0)`	Use sleep() – will sleep the amount of time in seconds
`delay = 1.0`	Define a variable (define variables at the top of the code, just under import statements)
`sleep(delay)`	Use a variable with sleep()
`leds.user_num(2, False)`	Turn off an LED
Mission 3 Lesson 2 – Time and Motion (Objective 6)	

`leds.user_num(0, True)`
`leds.ls_num(0, True)`
`leds.prox_num(0, True)`
	Turn on a single LED, the three types on CodeBot:
	User LEDs (Red, middle of the bot)
	Line sensor LEDs (Green, across the front)
	Proximity sensor LEDs (one on each side)

`leds.user(16)`	Use a decimal number to turn on/off all user LEDs (decimal number is converted to binary to control LEDs)
`leds.user(0b10101010)`	Use a binary number to turn on/off all user LEDs (0b for binary, then 0=off, 1=on for each LED)
`leds.ls(0b11111)`	Use a binary number to turn on all line sensor LEDs
`leds.ls(0b00100)`	Use a binary number to turn on only the middle line sensor LED
Mission 3 Lesson 3 – Time and Motion (Objectives 7-8)	
`from botcore import *`	Import entire library – * is a wildcard, which means everything

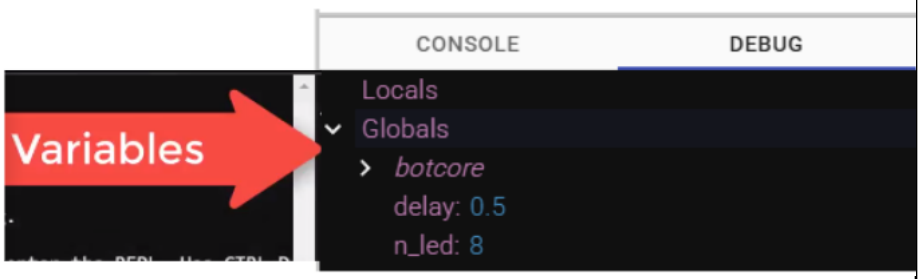
`motors.enable(True)`	Turn on motors – must be done before motors will turn and wheels move
`motors.run(LEFT, 50)` `motors.run(LEFT, -50)`	Turn left wheel forward at 50% power Turn left wheel backward at 50% power
`motors.enable(False)`	Turn off motors
`motors.run(LEFT, 30)` `motors.run(RIGHT, 30)`	Move forward in a straight line (Both motors have the same positive power)
`motors.run(LEFT, 30)` `motors.run(RIGHT, -30)`	Rotate the 'bot – motors have the same but opposite power Clockwise: left +, right - Counterclockwise: left -, right +
Mission 3 Lesson 4 – Time and Motion (Objectives 9-11)	
`buttons.was_pressed(0)`	Checks to see if BTN-0 was pressed; returns True or False

`leds.user(0b00011000)`
`sleep(10)`
`leds.user(0)`
`if buttons.was_pressed(0):`
`  {indented code}`
	Safety feature in code that only executes indented code if BTN-0 was pressed.

`if buttons.was_pressed(0):`
`  # Make a square`
`elif buttons.was_pressed(1):`
`  # Light show`
`else:`
`  # Stop`
	Example of control flow, or branching, using the if/elif/else structure

Unit 2 CodeBot Python Code By Mission

Mission 4 Lesson 1 – Animatronics (Objectives 1-3)	
`while True:`	Infinite loop
`leds.user_num(n_led, True)`	Use a variable for the LED number
`n_led = n_led + 1`	. Updating a variable; incrementing

Use the debugger to view variables. Open the console panel and click on the “debug” window while debugging.	
`if n_led == 8:` `    n_led = 0`	Check if a variable is the same as a specific value. If so, reset the variable to its initial value.
Mission 4 Lesson 2 – Animatronics (Objectives 4-7)	
`break`	Break out of a loop
`leds.ls_num(n_guests, True)`	Turn on a line sensor LED, using a variable to indicate which LED
`n_guests = n_guests + 1`	Increment
`spkr.pitch(440)`	Play a tone on the speaker. The argument is the pitch frequency in Hertz.
`spkr.off()`	Turn off the speaker
`buttons.was_pressed(0)`	Debounce a button press by resetting the internal status
Mission 4 Lesson 3 – Animatronics (Objectives 8-11)	
`while f < 1000:`	Basic structure of a while loop. In this example, the loop will continue to execute until f is equal to or greater than 1000.

`count = 0`
`while count < 10:`
`    count = count + 1`
	A while loop starts a block of code, so all commands that are to be repeated must be indented below the while loop.
	** The control variable (count) must be initialized before the loop starts.
	** The control variable (count) must be incremented inside the loop! Otherwise it is an infinite loop.

`from random import randrange`	Import the randrange() function from the random module (library)
`f = randrange(100, 1000)`	Generate a random integer within a range (possible numbers include first value up to one less than second value)
`def flashLEDs():` `    {indented block of code}`	Define a function. A function definition always has (), even if there are no parameters.

`def note(freq, duration):`
`  spkr.pitch(freq)`
`  sleep(duration)`
`  spkr.off()`
`  sleep(0.05)`	

`flashLEDs()` `note(349, 0.4)`	Function call. A function call uses the name and (), but not the “def”. If the function has parameters, the function call includes arguments, or values, that get passed to the parameters.
Mission 4 Lesson 4 – Animatronics (Objective 12)	
`F4 = 349` `C5 = 523`	Define a constant (usually with ALL CAPS)
`note(F4, 0.1)` `note(C5, 0.8)`	Use a constant as an argument in a function call

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.

Unit 4 CodeBot Python Code By Mission

Mission 7 Lesson 1 - Hot Pursuit (Objectives 1-3)	
`p = prox.detect()`	Calling this function pulses the emitter and detects reflected IR light. It returns a tuple of two bool values (left, right).
`p = prox.detect()` `leds.prox(p)`	Turn on the LED below each sensor if an object is detected. Example, if p = (True, False), the left LED is turned on and the right LED stays off.
`p = prox.detect(power, thresh)` `leds.prox(p)`	Proximity detection with optional parameters (power, threshold). The power parameter is a range from 1 to 8 for the brightness of the IR. The threshold is the detection sensitivity, with a range from 0%-100%.
`sensed = prox.range(10, power)`	This function scans multiple sensitivity levels to find the lowest detection threshold (0-100). It has four optional parameters: num_samples, power, range_low, range_high. We use the first two. 10 is the highest num_sample.
Mission 7 Lesson 2 - Hot Pursuit (Objectives 4-7)	
`if sensed[LEFT] > 0:` `    det = sensed[LEFT]`	After reading the proximity sensors (sensed), check only the LEFT sensor to see if it detected a reflection. If so, assign its sensitivity value to a variable.
`det = min(det, sensed[RIGHT])`	The math function min() finds the lowest, or minimum, value of the arguments. In this example, it selects the lowest value of either the current det or the sensitivity value of the RIGHT sensor.

`if det > 100:`
`    thresh = 100`
`else:`
`    thresh = det - 5`
	Bug fix to make sure thresh is the correct value.

`global thresh` `global power`	Keeps a global variable as global, even when it is assigned a value inside a function.

`while power < 9:`
`    cal_thresh()`
`    if thresh < 100:`
`        break`
`    power = power + 1`
	Loop that cycles through the range of powers to determine the best power level for the proximity sensors.

`print("Power=", power, "thresh=", thresh)`	Print statement with multiple arguments.
Mission 7 Lesson 3 - Hot Pursuit (Objectives 8-11)	

`if p[LEFT] and p[RIGHT]:`
`    motors.run(LEFT, 40)`
`    motors.run(RIGHT, 40)`
	Control the motors if both proximity sensors detect an object.

`elif p[LEFT]:`
`    motors.run(LEFT, 0)`
`    motors.run(RIGHT, 20)`
`elif p[RIGHT]:`
`    motors.run(LEFT, 20)`
`    motors.run(RIGHT, 0)`
	Turn the ‘bot either left or right if only the LEFT or RIGHT proximity sensor detects an object.

`go_motors  = False`	Define a Boolean variable that will toggle the motors enabled on/off.
`if buttons.was_pressed(0):` `    go_motors = not go_motors`	Use the “not” logical operator to toggle the value of the Boolean variable. It will change from True to False or False to True.
`motors.enable(go_motors)` `leds.user_num(go_motors)`	Use a Boolean toggle variable to turn on/off the motors and turn on/off a user LED.
Mission 9 Lesson 1 - All Systems Go!	
`system.pwr_volts()`	Returns a float (decimal number) for the current voltage, either from USB or batteries.
`system.pwr_is_usb()`	Returns an integer 0 if the power switch is set to batteries and 1 if USB.
`leds.user(15)`	Turn on the first four user LEDs so the battery is under load.

`leds.user(0)`	Turn off all user LEDs.
`pct = (v / 2) - 2`	Use the equation of a line to calculate the percentage, given the volts.
`leds.pwr(True)` `leds.pwr(False)`	Turn on the LED indicator for power. Turn off the LED indicator for power.
Mission 9 Lesson 2 - All Systems Go!	
`bot_temp = system.temp_C()`	Measure temperature in Celsius.
`bot_temp = system.temp_F()`	Measure temperature in Fahrenheit.
`sleep_ms(200)`	Delay program execution for 200 milliseconds.
`samples = []`	Initialize an empty list.
`samples.append(bot_temp)`	Append (add) a new item to a list.

`while i < count:`
`  sum = sum + nlist[i]`
`  i = i + 1`
	Traverse a list using a loop to sum all the items.

`return sum / count`	Return the average without assigning the value to a variable.
`samples.clear()`	Clear a list of all items.
`BASELINE = 25.5` `DEADBAND = 3.0`	Define a constant.
`for i in range(5):`	A quick way to loop a block of code five times.
`if t > BASELINE + DEADBAND:` `  leds.user(0b11111111)`	Compare temperature for a value that is too high, above the baseline + deadband.
`elif t < BASELINE - DEADBAND:` `  leds.ls(0b11111)`	Compare temperature for a value that is too low, below the baseline - deadband.

Mission 9 Lesson 3 - All Systems Go!	
`accel.dump_axes()`	Prints the 3-axis values to the console
`x, y, z = accel.read()`	Reads the current axis values and returns a tuple of integers, ranging from -32767 to +32768
`now = accel.read()`	Read the accelerometer and assign all three values to a tuple.
`now[0]`	Access the X value of the accelerometer reading.
`before = now`	Assign the same tuple (now) to a new variable (before).
`dx = now[0] - before[0]`	Calculate the difference between current reading and previous reading.
`if abs(dx) > SENS:` `    alarm()`	If the difference between readings is more than the sensitivity, sound an alarm.