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)`
	`leds.ls_num(0, True)`
	`leds.prox_num(0, True)`
	User LEDs (middle of the bot)
	Line sensor LEDs (across the front)
	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 `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 `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):` `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	 Open the console panel while debugging

Reset a variable to stay within a range	`n_led = n_led + 1`
	`if n_led == 8:`
	`    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)` `sleep(0.1)` the (argument) is the pitch frequency
Turn off the speaker	`spkr.off()`
Debounce a button press	`buttons.was_pressed(0)`

While loop	`while count < 10:` (will iterate, or repeat, 10 times if count starts at 0)
Import random library	`from random import randrange`
Get a random number within a range	`f = randrange(100, 1000)`

Define a function	`def flashLEDs():`
	`  leds.user(0b11111111)`
	`  sleep(0.5)`
	`  leds.user(0b00000000)`
	`  sleep(0.5)`
	
	`# Function to play a note`
	`def note(freq, duration):`
	`  spkr.pitch(freq)`
	`  sleep(duration)`
	`  spkr.off()`
	`  sleep(0.05)`

Call a function	`flashLEDs()` `note(F4, 0.4)`
Mission 5 - Fence Patrol	
Read a line sensor	`ls.read(num)  # Sensor 'num' can be 0, 1, 2, 3, or 4` `val = ls.read(n)` (returns a value between 0 and 4095)
Display the value of a variable in the console	`print(val)` `print("Line sensor value = ", val)`

Assign a Boolean result of a comparison to a variable
 	
Use the Boolean variable in code
	`threshold = 2500`
	`is_detected = val < threshold`
	`leds.ls_num(0, is_detected)`

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

Use a comparison with a while loop and use the control variable as an argument in a function call	`n = 0`
	`while n < 5:`
	`  detect_line(n)`
	`  n = n + 1`

Wait loop
(safe driving)
	`while True:`
	`  if buttons.was_pressed(0):`
	`  break`

Return statement	`return is_detected` `return got_line`

Call to a function that has a return	`hit = scan_lines()` `if detect_line(count):`
Use a variable to turn on LEDs	`leds.user(line_count)` line_count will be from 0 to 255

Wrap-around the line_count variable for binary numbers	`line_count = line_count + 1`
	`if line_count == 256:`
	`    line_count = 0`

Mission 6 - Line Follower	
Create a list	`detected = [False, False, False, False, False]`
Update a specific value in a list	`detected[count] = val > thresh`

Use a list with LEDs	`leds.ls([False, True, True, True, False])`
	`vals = check_lines(threshold)`
	`leds.ls(vals)`

Botcore line sensors function (similar to check_lines) but faster	`vals = ls.check(thresh, is_reflective)` `leds.ls(vals)` ls.check() takes 2 parameters It has a second parameter `is_reflective` that controls whether **"detected"** means the **sensor is** `> thresh` or `< thresh`. It returns a tuple rather than a list.
Using or (logical operator)	`elif vals[1] or vals[2] or vals[3]:` can have two or more conditions; if any of the conditions are true, the statement will evaluate to true
Comparing with a tuple	`elif vals == (0,1,1,0,0):`
Code needed to change a global variable inside a function	`global count` `global thresh, is_reflective`
Built-in math operations	`abs(x)` `round(x, ndigits)`
Mission 7 - Hot Pursuit	
Read the proximity sensors	`prox.detect()` returns a tuple (left, right) with values True or False

	`vals = prox.detect()`
	`left_detected = vals[0]`
	`right_detected = vals[1]` Index values: 0 = left 1 = right

Proximity LEDs	`# Check proximity sensors`
	`p = prox.detect()`
	
	`# Show (left, right) on the PROX LEDs`
	`leds.prox(p)`

Use parameters	**`P = prox.detect(power, threshold)`** Power is the "'bot flashlight" with settings from 1 to 8 (high power) 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()` `prox.range(num_samples, power, range_low, range_high)` 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`
	`go_motors = False`
	
	`go_motors = not go_motors # (not False) == True`
	
	`go_motors = not go_motors # (not True) == False`

Mission 8 - Navigation	
Read a wheel encoder	`# Read the encoder's analog value.` `val = enc.read(side)  # 'side' is LEFT or RIGHT`

Compare a state; Uses the != (not equals) comparison operator	`slot = sense_slot()  # Read current state`
	`if enc_state != slot:  # Compare to previous state`
	`  # Disc has moved...`

Define a list of counters, initializing to 0	`# Start with zero counts for LEFT and RIGHT.` `enc_count = [0, 0]`

Increment a list of counters	`# Increment the LEFT and RIGHT counts.`
	`enc_count[LEFT] = enc_count[LEFT] + 1`
	`enc_count[RIGHT] = enc_count[RIGHT] + 1`

Constant for pi from the math module	`import math`
	`WHEEL_DIA = 66.5`
	`WHEEL_CIRC = (math.pi * WHEEL_DIA)`

Copy a list (not make a reference)	`# Use the copy() method!` `start_count = enc_count.copy()`
Make a new variable that will reference the same list	`# This will NOT make a new list!` `start_count = enc_count`

Debounce the button press	`while True:`
	`  # Wait for BTN-0. Good robot.`
	`  buttons.was_pressed(0)  # debounce`
	`  while True:`
	`  if buttons.was_pressed(0):`
	`  break`

Marks the passage of time in milliseconds; the counter starts at 0 when the device boots and keeps count up while it is running.
 	
Diff gives the difference between start and stop
	`import time`
	
	`t_start = time.ticks_ms()`
	`# Do some stuff that takes time...`
	`t_stop = time.ticks_ms()`
	
	`t_diff = time.ticks_diff(t_stop, t_start)`
	`print("That took ", t_diff, " milliseconds!")`

Feedback loop
 	
Slower than desired gives a positive err, increase power
	`# Calculate: err = (Input - Output) * Fpwr`
	`err = ( target_speed - cur_speed[LEFT] ) * FEEDBACK_PWR`
	
	`# Apply feedback to System (adjust motor power)`
	`power[LEFT] = power[LEFT] + err`
	`motors.run(LEFT, power[LEFT])`

Delay in milliseconds	`sleep_ms()`
Set a default parameter	`def drive(cm, speed, dir=[+1, +1]):`

Mission 9 - All Systems Go!	

Built-in function that measures power supply voltage (battery or USB)	`# Measure power supply voltage (battery or USB)`
	`v = system.pwr_volts()`
	Returns the float power supply voltage; can come from USB or battery pack, depending on power switch

Returns an integer for which power source is being used.	`# Am I powered by USB or Battery?` `on_usb = system.pwr_is_usb()` Returns 1 for USB, 0 for battery pack
Turn on power LED	`leds.pwr(True)` activates the red LED just above the power switch
Add a value to a list	`samples.append(temperature)`
Empty a list	`samples.clear()`
Read the accelerometer	`x, y, z = accel.read()` `now = accel.read()` returns a tuple: 3 integers from -32767 to + 32768
Prints the accelerometer reading on the console	`accel.dump_axes()`
Calculate the difference between current reading and previous reading	`dx = now[0] - before[0]`
If the difference between readings is more than the sensitivity, sound alarm	`if abs(dx) > SENS:` `    alarm()`