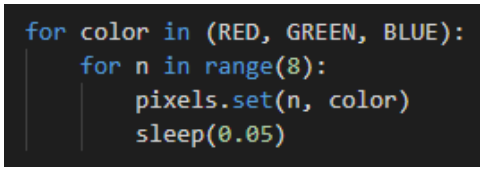
CodeAIR Python Code By Mission

Mission 2 – Introducing CodeAIR	
`from codeair import *`	Import the codeair library; all built-in code specific to CodeAIR
`leds.set(num, brightness)` `leds.set(0, 50)`	Sets the user LED at a brightness level. In this example, LED 0 is set to 50 percent brightness
Mission 3 – Pre-Flight Check	
`from time import sleep`	Import the time library to access built-in timing functions like sleep
`leds.set(0, 0)`	Turn off an led; use a brightness of 0

`leds.set(0, 50)`
`sleep(1)`
`leds.set(0, 0)`
`sleep(1)`
	Blink an LED for 1 second intervals.

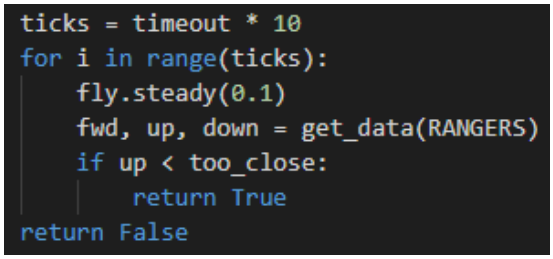
`while True:`	Infinite loop (instruction ends with a colon (:) and block underneath is indented)
`speaker.beep(frequency, duration)` `speaker.beep(440, 200)`	Play a note (or sound) using CodeAIR's speaker In this example, the frequency is 400 and the duration is 200 ms
`D5 = 587`	Constant definition
`leds.set_status(50)`	A single LED positioned near the USB connector. The command needs a single argument for brightness.
`COLOR_LIST = (BLACK, BROWN, RED, ORANGE, YELLOW, GREEN, BLUE, PURPLE, GRAY, WHITE, CYAN, MAGENTA, PINK, LIGHT_GRAY, DARK_GREEN, DARK_BLUE)`	Standard color definitions that are included in the **codeair** library from the **colors** module.
`pixels.set(num, color)` `pixels.set(0, RED)`	Set a pixel LED to a specific color In this example, pixel 0 is set to RED
`pixels.set(0, BLACK)`	Turn off a pixel LED. Here, color names are in ALL CAPS because they are included in the pre-defined COLOR_LIST.
`for n in range(8):`	For loop that starts at 0 and goes up to but not including the ending value. In this example, the iteration would be 0, 1, 2, 3, 4, 5, 6 and 7.
	Loop for turning pixels red, then green, then blue.
`pixels.set(TOP_FRONT_LEFT, RED)`	Pixels can be designated with a number or constant for position: BOTTOM_FRONT_LEFT, BOTTOM_FRONT_RIGHT, BOTTOM_REAR_LEFT, BOTTOM_REAR_RIGHT, TOP_FRONT_LEFT, TOP_FRONT_RIGHT, TOP_REAR_RIGHT, BOTTOM_REAR_RIGHT
`pixels.fill(WHITE, brightness=50)`	Turns all 8 pixels WHITE at brightness 50. This code is much shorter than turning on all 8 pixels individually.

`sleep(1.0)`
`pixels.fill(WHITE, brightness=50)`
`sleep(0.02)`
	Strobe

Mission 4 – Flight Safety	
`buttons.was_pressed(BTN_0)`	Checks to see if B0 was pressed since the last check.
`break`	Breaks out of the nearest enclosing loop
`if buttons.was_pressed(BTN_0):` `  break`	If statement (branching) that checks for a button press. buttons.was_pressed(BTN_0) is either True or False.

`while True:`
`  if buttons.was_pressed(BTN_0):`
`  break`
	If statement in an infinite loop. The code waits for a button press before moving to the next line of code.

`pixels.fill(YELLOW)`	Sets all 8 pixels to YELLOW (built-in color)
`pixels.off()`	Turn off all 8 pixels
`sleep(0.1)` `buttons.was_pressed()`	Debounce the buttons. This line of code resets both buttons!
`from flight import *`	Imports the flight module so you can use built-in functions, like motor_test()
`motor_test(True)` `motor_test(False)`	Start / stop a motor test that spins the motors but not fast enough to lift off.
`def button_arm():`	Function definition. The indented block below is the code of the function. A function definition always has () for parameters, even if none are given.
`return do_launch`	Returns (sends) data from the function back to the code that called it. A return ends the function.
`if button_arm():`	Call the function button_arm(), which returns a True or False value
`set_param('motorPowerSet.m2', 30000)`	Set motor (m2) power (30000)
`set_param('motorPowerSet.enable', 1)`	Enable power to the motors
`set_param('motorPowerSet.enable', 0)`	Disable power to the motors
Mission 5 - Hovering Flight	
`'''This is a docstring'''`	Document string that should go at the top of any module
`fly.take_off(height_meters)`	Ascend to given height altitude
`fly.steady(seconds)`	Hover, allows code to pause while keeping the flight controller running
`fly.land()`	Descend to the floor
`fly.forward(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.back(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.left(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)

`fly.right(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.up(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.down(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`get_data(RANGERS)`	Returns the (forward, up, down) distance in millimeters
`if up < too_close:` `  # sound alarm`	If statement with a condition
`fwd, up, down = get_data(RANGERS)`	Unpack the data from the rangers from the three values in the tuple to three variables
	Algorithm for polling with a blocking function. In this example, timeout is a parameter that receives seconds from an argument. The polling will happen ten times per second.
`speaker.beep(400, 0)`	Causes the beep to play continuously. Requires speaker.off() to stop the beep.
`count = count + 1`	Incrementing or updating a variable
`leds.set_mask(0, 0)`	Turn off all the blue LEDs
`fly.start_forward()`	Non-blocking function that starts moving forward at the default velocity and returns immediately so the next instruction can be executed
`fly.stop()`	Stop any motion and hover
`fly.turn_left(degrees)`	A blocking function that turns the drone degrees left
`if count == 8:`	Checks if **count** is the same as 8. If it is, a branch of code is executed.
Mission 6 - Navigate	
`dx, dy = get_data(FLOW)`	Read data from the flow sensor; returns the change in x direction and change in y direction
`print(x, y)`	A simple print statement that converts data to strings and displays them on the console
`print("Flow Sensor Output")`	Print a string text on the console
`print(f"x={x}, y={y}")`	F-string with replacement fields in curly braces
`abs(x)`	Returns the absolute value of x
`vbatt = power.battery_voltage(10)`	Read battery voltage, average 10 samples
`amps = power.charger_current()`	Read charging current when connected with USB
`usb_connected = power.is_usb()`	Returns True if currently powered by USB

`value = 0b1001`	Set the value to 9 using binary
`leds.set_mask(255, 50)`	Set BYTE LEDs to 255 (on) with brightness = 50
`leds.set_mask(0b10101010, 50)`	Set BYTE LEDs using binary
`If __name__ == '__main__':`	Detects when this program is being run as the “main program” instead of an import
`try:`	A block of code that executes when no error occurs, or until an error occurs.
`except:`	A block of code that lets your program respond to an error without crashing.