The purpose of this document is to provide a high level overview of the SB2 API's functions for quick reference. Modules need to be included in the code using the **import** statement. Classes can be accessed by appending the module with the class name using a period. Methods and functions can be accessed the same way. The follow program demonstrates the use of modules, classes, and functions to turn on the red (1) and green(2) LEDs:

```
import pyb
redLED=pyb.LED(1)
redLED.on()
pyb.LED(2).on()
```

Be mindful of both spelling and capitalization; the names of the class, methods, functions, and objects must be used exactly as seen below. Python ignores whitespaces within lines of code, so for example if in the program above you replaced the second line with "`redLED = pyb.LED( 1 )`" it would function exactly the same. The names of objects however must be kept in tact, that is, no spaces. The line "`red LED=py b.LE D(1)`" would not work.

(Module) - **pyb**

(Class) - **LED(i)** - Used to access the tri-color LED. The parameter i can be either 1, 2, or 3 to access the red, green, and blue LEDs, respectively.

(Method) - **on()** - Turns the LED on

(Method) - **off()** - Turns the LED off

(Module) - **switch**

(Class) - **Switch(s)** - Used to access the five programmable buttons on the SB2. Allowed values for parameter s are 'a', 'b', 'c', 'up', and 'down', to access the respective buttons.

(Method) - **value()** - Returns True if the associated button is pressed, and False otherwise

(Module) - **color**

(Function) - **RGB(r, g, b)** - Returns a color object that can be used with the graphics module. The parameters are integer values of 0-255 which represent the amount of red, green, and blue in the returned color.

(Object) - **RED, GREEN, BLUE, WHITE, BLACK** - Predefined colors

(Module) - **graphics**

(Function) - **paint(color)** - Fills the LCD with a given color. Colors can be created using the RGB() function of the color class
(Function) - **print(string, x, y, color)** - Prints the given string to the LCD starting at the x,y coordinate. The new text will be the given color.
(Function) - **printTop(string, color)** - Prints the given string at the top of the LCD with the given color.
(Function) - **printMiddle(string, color)** - Prints the given string to the middle of the LCD with the given color.
(Function) - **printBottom(string, color)** - Prints the given string to the bottom of the LCD with the given color.
(Function) - **erase(i, x, y, o)** - Fills the space starting x pixels from the left of the screen and y pixels from the top of the screen the given color o. The integer i is the number of characters that will be replaced with the color (used to “erase” text)

(Function) - **line(x1, y1, x2, y2, color)** - Draws a line with the given color from x1,y1 to x2,y2. THe line has a width of 1 pixel.
(Function) - **rectangle(x, y, width, height, color)** - Draws a rectangle starting at coordinate x,y with a width and height of w and h. The edges of the rectangle are 1 pixel wide and the color o.
(Function) - **fill_rectangle(x, y, width, height, color)** - Draws a rectangle starting at coordinate x, y with a given width and height. The shape is filled in with the given color.
(Function) - **circle(x, y, r, color)** - Draws a circle centered around coordinate x, y with a given radius r. The circle will be drawn with the given color and have a thickness of 1 pixel.
(Function) - **fill_circle(x, y, r, color)** - Draws a circle centered around coordinate x, y with a radius r. THe circle will be filled in with the given color.
(Function) - **triangle(x1, y1, x2, y2, x3, y3, color)** - Draws a triangle with the three points at the given coordinates. The triangle is drawn with the given color and has an edge thickness of 1 pixel.
(Function) - **fill_triangle(x1, y1, x2, y2, x3, y3, color)** - Draws a triangle with the three points at the given coordinates. The triangle is filled in with the given color.

(Function) - **arrow(angle, x, y, length, color)** - Draws an arrow with a given color pointing to a given x,y coordinate with a given length. The function accepts angles in 45 degree increments (i.e., 0, 45, 90, 135, etc.).

(Object) - **lcdheight** - The height of the LCD in pixels.
(Object) - **lcdwidth** - The width of the LCD in pixels.

(Module) - **utime**

(Function) - **sleep(s)** - Delay of s seconds
(Function) - **sleep_ms(s)** - Delay of s milliseconds
(Function) - **sleep_us(s)** - Delay of s microseconds

(Module) - **sb**

(Class) - **Motor(x)** - Used for interacting with the motors. Accepts either 1 or 2 as a parameter.

(Method) - **sleep(bool)** - Forces the associated motor to enter or exit sleep mode. True sets the motor to sleep and False wakes it up. Leaving the parameter blank returns either True or False depending on the sleep state.
(Method) - **brake_mode(bool)** - Sets the motor brake mode. If true, the motor will “brake” at the end of movement, otherwise the motor will be allowed to roll. Leaving the parenthesis blank will return either True or False depending on the state of the brake mode.
(Method) - **speed(x)** - Sets the stepper motor to move continuously. Values above 500 may cause the motor to stall. Leaving the parenthesis blank will return the current speed.
(Method) - **distance(steps, speed, acceleration, blocking)** - Causes the motor shaft to rotate a given number of steps at a certain speed. Speeds above 500 may cause the motor to stall. Acceleration is the time it takes to wind up to the given speed (a value of 10 is recommended for most purposes). Setting blocking to True prevents the next line of code from being executed until the motor finishes moving.
(Method) - **stop()** - Causes the motor to stop moving.

(Function) - **setWheelDiameter(x)** - Sets the wheel diameter in millimeters. This value is used for distance conversions. Default is 85.
(Function) - **getWheelDiameter()** - Returns the current wheel diameter in millimeters.
(Function) - **inches_to_steps(x)** - Converts the given number of inches to steps based on the wheel’s diameter. Used for the distance() method.
(Function) - **mm_to_steps(x)** - Converts the given number of millimeters to steps based on the wheel’s diameter. Used for the distance() method.
(Function - **decirevs_to_steps(x)** - Converts the given number of decirevs to steps. Used for the distance() method.
(Function) - **steps_to_inches(x)** - Converts a given number of steps to inches based on the wheel’s diameter.
(Function) - **steps_to_mm(x)** - Converts the given number of steps to millimeters based on the wheel’s diameter.
(Function) - **steps_to_decirevs(x)** - Converts the given number of steps to decirevs.

(Class) - **Servo(x)** - Used for programming the servo ports.

(Method) - **pulse_width(x)** - Sets the pulse width of the servo signal to the given number of microseconds

(Module) - **remote**

(Class) - **Remote()** - Used to access remote controller methods
(Method) - **pair()** - Pairs the SB2 with a remote controler
(Method) - **read(RemoteData, timeout)** - Reads the current radio data and puts the information into a RemoteData object. If data cannot be read before the timeout, nothing happens.

(Class) - **RemoteData()** - Used to store remote controller data
(Object) - **buttons** - A length 15 bytearray object that stores data on which remote controller buttons are being pressed. Used with bitwise operators and the "_BIT" objects to determine the state of single buttons.
(Object) - **ljoy_up_down** - Contains a numerical value representing the up/down position of the left joystick.
(Object) - **ljoy_left_right** - Contains a numerical value representing the left/right position of the left joystick.
(Object) - **rjoy_up_down** - Contains a numerical value representing the up/down position of the right joystick.
(Object) - **rjoy_left_right** - Contains a numerical value representing the left/right position of the right joystick.
(Object) - **SLCT_BIT**, **STRT_BIT** - Numerical value representing the select and start buttons. To be compared with the RemoteData().buttons object.
(Object) - **L1_BIT**, **L2_BIT**, **L3_BIT**, **R1_BIT**, **R2_BIT**, **R3_BIT** - Numerical value representing the left and right buttons (L3 and R3 are the joystick buttons). Used in conjunction with the RemoteData().buttons object.
(Object) - **DPAD_UP_BIT**, **DPAD_RT_BIT**, **DPAD_DN_BIT**, **DPAD_LT_BIT** - Numerical value representing the buttons on the directional pad. Used in conjunction with the RemoteData().buttons object.
(Object) - **A_BUTTON**, **B_BUTTON**, **X_BUTTON**, **Y_BUTTON** - Numerical values representing the colored buttons. Used in conjunction with the RemoteData().buttons object.

(Module) - **pnp**

(Class) - **GPIO(position)** - Used to access the UEXT port as pins (also for the LED plug and play modules). Accepts one parameter, either “top”, “left”, or “right”.

(Method) - **togglePin(pin)** - Toggles the given pin between on and off states.
(Method) - **allOn()** - Turns on all of the pins.
(Method) - **allOff()** - Turns off all of the pins.

(Class) - **SevenSegmentDisplay(position)** - Used to access the methods of the seven segment display (SSD) plug and play module. Accepts one parameter, either “top”, “left”, or “right”.

(Method) - **clear()** - Turns off all of the LEDs on the SSD.
(Method) - **displayNumber(x)** - Displays a given number. Only integers from 0 to 9 are allowed.
(Method) - **toggleDP()** - Toggles the decimal point.

(Class) - **SHT21(position)** - Used to access the methods of the SHT21 temperature/humidity sensor plug and play module. Accepts one parameter, either “top”, “left”, or “right”.

(Method) - **getTempC()** - Returns the current temperature in degrees Celsius.
(Method) - **getTempF()** - Returns the current temperature in degrees Fahrenheit.
(Method) - **getRH()** - Returns the relative humidity in percent.

(Class) - **MPU6050(position)** - Used to access the methods of the MPU6050 accelerometer/gyroscope plug and play module. Accepts one parameter, either “top”, “left”, or “right”.

(Method) - **getAcceleration()** - Returns x, y, and z axis acceleration data in meters per second squared.
(Method) - **getAngularVelocity()** - Returns angular velocity about the x, y, and z axes. Data is returned in units of degrees per second.

(Class) - **OPT3001(position)** - Used to access the method of the OPT3001 plug and play module. Accepts one parameter, either “top”, “left”, or “right”.

(Method) - **getLux()** - Returns the ambient brightness in Lux.