```
#include <FastLED.h>
FASTLED_USING_NAMESPACE

#if defined(FASTLED_VERSION) && (FASTLED_VERSION < 3001000)

#warning "Requires FastLED 3.1 or later; check github for latest code."

#endif

#define DATA_PIN    3

//#define CLK_PIN   4

#define LED_TYPE    WS2811

#define COLOR_ORDER GRB

#define NUM_LEDS    300

CRGB leds[NUM_LEDS];

#define BRIGHTNESS         96

#define FRAMES_PER_SECOND  120

bool gReverseDirection = false;

void setup() {

  delay(3000); // 3 second delay for recovery

  FastLED.addLeds<LED_TYPE,DATA_PIN,COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);

  FastLED.setBrightness(BRIGHTNESS);

}


typedef void (*SimplePatternList[])();

SimplePatternList gPatterns = { rainbow, rainbowWithGlitter, confetti, sinelon, juggle, bpm, Fire2012, Fire2012WithPalette,
whitesparkle };


uint8_t gCurrentPatternNumber = 0; // Index number of which pattern is current

uint8_t gHue = 0; // rotating "base color" used by many of the patterns
```

```
void loop()
{
  gPatterns[gCurrentPatternNumber]();
  FastLED.show();
  FastLED.delay(1000/FRAMES_PER_SECOND);
  EVERY_N_MILLISECONDS( 20 ) { gHue++; } // slowly cycle the "base color" through the rainbow
  EVERY_N_SECONDS( 10 ) { nextPattern(); } // change patterns periodically
}
#define ARRAY_SIZE(A) (sizeof(A) / sizeof((A)[0]))
void nextPattern()
{
  gCurrentPatternNumber = (gCurrentPatternNumber + 1) % ARRAY_SIZE( gPatterns);
}
void rainbow()
{
  fill_rainbow( leds, NUM_LEDS, gHue, 7);
}
void rainbowWithGlitter()
{
  rainbow();
  addGlitter(80);
}
void addGlitter( fract8 chanceOfGlitter)
{
  if( random8() < chanceOfGlitter) {
```

```
    leds[ random16(NUM_LEDS) ] += CRGB::White;
  }
}

void confetti()
{
  fadeToBlackBy( leds, NUM_LEDS, 10);
  int pos = random16(NUM_LEDS);
  leds[pos] += CHSV( gHue + random8(64), 200, 255);
}
void sinelon()
{
  fadeToBlackBy( leds, NUM_LEDS, 20);
  int pos = beatsin16( 13, 0, NUM_LEDS-1 );
  leds[pos] += CHSV( gHue, 255, 192);
}
void bpm()
{
  uint8_t BeatsPerMinute = 62;
  CRGBPalette16 palette = PartyColors_p;
  uint8_t beat = beatsin8( BeatsPerMinute, 64, 255);
  for( int i = 0; i < NUM_LEDS; i++) { //9948
    leds[i] = ColorFromPalette(palette, gHue+(i*2), beat-gHue+(i*10));
  }
}
void juggle() {
```

```
 fadeToBlackBy( leds, NUM_LEDS, 20);
 byte dothue = 0;
 for( int i = 0; i < 8; i++) {
  leds[beatsin16( i+7, 0, NUM_LEDS-1 )] |= CHSV(dothue, 200, 255);
  dothue += 32;
 }
}

#define COOLING  55
#define SPARKING 120

void Fire2012()
{
 static byte heat[NUM_LEDS];

  for( int i = 0; i < NUM_LEDS; i++) {
    heat[i] = qsub8( heat[i],  random8(0, ((COOLING * 10) / NUM_LEDS) + 2));
  }

  for( int k= NUM_LEDS - 1; k >= 2; k--) {
    heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2] ) / 3;
  }

  if( random8() < SPARKING ) {
    int y = random8(7);
    heat[y] = qadd8( heat[y], random8(160,255) );
```

```
    }


    // Step 4.  Map from heat cells to LED colors
    for( int j = 0; j < NUM_LEDS; j++) {
      CRGB color = HeatColor( heat[j]);
      int pixelnumber;
      if( gReverseDirection ) {
        pixelnumber = (NUM_LEDS-1) - j;
      } else {
        pixelnumber = j;
      }
      leds[pixelnumber] = color;
    }
}


CRGBPalette16 gPal;
void Fire2012WithPalette()
{
  static byte heat[NUM_LEDS];
    for( int i = 0; i < NUM_LEDS; i++) {
      heat[i] = qsub8( heat[i],  random8(0, ((COOLING * 10) / NUM_LEDS) + 2));
    }
    for( int k= NUM_LEDS - 1; k >= 2; k--) {
      heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2] ) / 3;
    }
    if( random8() < SPARKING ) {
```

```
    int y = random8(7);
    heat[y] = qadd8( heat[y], random8(160,255) );
   }
   for( int j = 0; j < NUM_LEDS; j++) {
    byte colorindex = scale8( heat[j], 240);
    CRGB color = ColorFromPalette( gPal, colorindex);
    int pixelnumber;
    if( gReverseDirection ) {
     pixelnumber = (NUM_LEDS-1) - j;
    } else {
     pixelnumber = j;
    }
    leds[pixelnumber] = color;
   }
}

void whitesparkle() {
  for(int whiteLed = 0; whiteLed < NUM_LEDS; whiteLed = whiteLed + 1) {
    leds[whiteLed] = CRGB::White;
    FastLED.show();
    delay(10);
    leds[whiteLed] = CRGB::Black;
  }
}
```