ESERCIZIO 1:

```
void blinkGreen(){
  greenLedState = !greenLedState;
  digitalWrite(GLED_PIN, greenLedState);
}

void setup() {
  // put your setup code here, to run once:
  pinMode(RLED_PIN, OUTPUT);
  pinMode(GLED_PIN, OUTPUT);
  Timer1.initialize(G_HALF_PERIOD *1e06);
  Timer1.attachInterrupt(blinkGreen);

}

void loop() {
  // put your main code here, to run repeatedly:
  redLedState = !redLedState;
  digitalWrite(RLED_PIN, redLedState);
  delay(R_HALF_PERIOD * 1e03);

}
```

Obiettivo: Si vuole utilizzare la Yùn Arduino per pilotare due LED, in modo da farli lampeggiare con periodi indipendenti tra loro.
Componenti hardware utilizzate: breadboard, led rosso, led verde, resistenza
Componenti software utilizzate: GPIO, interrupt
Sketch: I due led sono stati connessi ai pin 11 e 12 (led verde e led rosso rispettivamente), quindi definiamo i pin come costanti e li attiviamo in output nella funzione *setup()*; come indicato dal testo, saranno delle costanti definite a piori anche i due semiperiodi di lampeggiamento, cioè il tempo in cui ogni led rimarrà nel suo stato acceso o spento.
Il led rosso viene pilotato nella funzione *loop()*, ripetuta per definizione infinite volte, in cui semplicemente cambiamo lo stato del led (LOW=spento, HIGH=acceso); mettiamo la funzione in attesa con la funzione delay in modo da far permanere il led nello stato corrente per un semiperiodo.
I due periodi di lampeggiamento dovranno essere indipendenti, quindi non posso controllare quello del led verde dalla *loop()*; per farlo, si sceglie di usare una Interrupt Service Routine, cioè una funzione richiamata dal sistema al verificarsi di un evento, in questo caso lo scadere di un timer, definito nella libreria *TimerOne*. Questa libreria fornisce un metodo *initialize()* per far partire il countdown, a cui diamo il valore del semiperiodo del led verde, e un metodo *attachInterrupt()*, che richiama la mia ISR allo scadere di quel timer. Lei è la funzione *blinkGreen()*, che semplicemente cambia stato al led verde.

ESERCIZIO 2:

```
void blinkGreen(){
  greenLedState = !greenLedState;
  digitalWrite(GLED_PIN, greenLedState);
}

void serialPrintStatus(){
  if(Serial.available() > 0){
    int inByte = Serial.read();
    if(inByte == 'R'){
      if (redLedState == LOW){
        Serial.println("Led Rosso: spento");
      }
      else {
        Serial.println("Led Rosso: acceso");
      }
    }
    else if(inByte == 'L'){
      if (greenLedState == LOW){
        Serial.println("Led Verde: spento");
      }
      else {
        Serial.println("Led Verde: acceso");
      }
    }
```

Il secondo esercizio permette di avviare una comunicazione tramite una porta seriale.
Componenti hardware utilizzate: breadboard, led, resistenza e porta seriale.
Componenti software utilizzate: stesse dell'esercizio 1, più i metodi della libreria Serial, tra questi troviamo:

- Il metodo *begin()* che permette di creare una connessione seriale.
- Il metodo *available()* permette di verificare se ci sono byte disponibili nel serial buffer.
- Il metodo *read()* per leggere il contenuto del serial buffer.

A differenza del precedente esercizio, dovremmo essere in grado di leggere lo stato dei LED nel Serial Monitor. Precisamente dobbiamo essere in grado di ricevere un carattere che indica il LED tramite seriale e, successivamente, lo sketch deve inviare un messaggio al PC contenente lo stato del LED corrispondente.

```
    else {
      Serial.println("Carattere non riconosciuto");
    }
  }
}

void setup() {
  // put your setup code here, to run once:
  pinMode(RLED_PIN, OUTPUT);
  pinMode(GLED_PIN, OUTPUT);
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Lab 1.2 Starting");
  Timer1.initialize(G_HALF_PERIOD *1e06);
  Timer1.attachInterrupt(blinkGreen);


}

void loop() {
  // put your main code here, to run repeatedly:
  redLedState = !redLedState;
  digitalWrite(RLED_PIN, redLedState);
  serialPrintStatus();
  delay(R_HALF_PERIOD * 1e03);

}
```

Sketch

La *setup(),* oltre a settare i pin GPIO come OUTPUT, permette di:

1. creare una comunicazione seriale con il PC ad una determinata velocità;
2. attendere che venga stabilita la connessione e quindi non far partire il programma fino a quando non viene aperto il Serial Monitor;
3. stampa di inizializzazione;
4. inizializzare un timer e agganciare ad esso un interrupt. In questo modo viene gestita l'accensione e lo spegnimento del PIN verde.

La *loop()*, invece, ad ogni iterazione chiama la funzione *serialPrintStatus()*. Questa controlla se sono stati ricevuti dei caratteri e se questi corrispondono al codice ASCII "R" o "L". In caso affermativo, ritornano lo stato del LED corrispondente.
Dal momento che il LED rosso è gestito nella funzione *loop()* ci verrà comunicata l'accensione o lo spegnimento del LED qualche istante prima o dopo che il LED è stato acceso o spento. Questo "ritardo" è dovuto al fatto che il codice nella *loop()* è gestito in maniera sequenziale.

Qualificatore volatile

Permette di modificare il modo in cui il compilatore tratta le variabili. In dettaglio, indica al compilatore di non effettuare ottimizzazioni basate sui registri e, di conseguenza, garantisce una corretta interpretazione della variabile. È, quindi, di particolare importanza nel momento in cui la variabile è usata sia nell'interrupt che nella funzione *loop()*.

ESERCIZIO 3

```
void checkPresence(){
  int value = digitalRead(PIR_PIN);
  if (value == HIGH){
    Serial.println("Motion Detected");
    tot_count +=1;
    redLedState = HIGH;
    digitalWrite(RLED_PIN, redLedState);
  }
  else{
    redLedState = LOW;
    digitalWrite(RLED_PIN, redLedState);
  }

}
```

Obiettivo di questo esercizio era quello di rilevare e di contare dei movimenti.
Componenti hardware utilizzate: sensore di movimento e porta seriale.
Componenti software utilizzate: interrupt sul cambiamento dei valori del pin di GPIO.

Sketch

La *loop()* stampa il numero di eventi rilevati fino a quel momento.

La *setup()* permette di settare i pin, di avviare una comunicazione seriale con il PC e di collegare una ISR al verificarsi di un cambiamento di valore sul pin stesso. La *setup()*, inoltre, comprende la funzione *attachInterrupt()*, che richiede tre parametri:

1. *digitalPinToInterrupt(PIN)* ritorna, leggendo nell'Interrupt Vector Table, qual è il valore dell'interrupt corrispondente a quel pin;
2. ISR;
3. definisce quando l'interrupt deve essere "triggerato".

La *checkPresence()* permette di reagire ai cambi di valore sul pin tramite una Interrupt Service Routine, che grazie al parametro CHANGE, è attiva su entrambi i fronti. L'output digitale del sensore PIR deve essere collegato ad uno degli input digitali della Yùn e, in particolare, abbiamo usato uno dei pin GPIO che supportano Interrupt (pin D7).
Il valore letto dal pin D7 viene riprodotto sul LED, il quale risulterà acceso nel momento in cui il sensore produce un valore alto, oppure spento nella situazione opposta.
Inoltre, la ISR incrementa il numero di movimenti ogni qual volta vengono rilevati.

PIR Motion Sensor: HC-SR501

Il sensore consente di individuare il movimento tramite tecnologia PIR e ha una serie di parametri configurabili manualmente, tra i quali:

1. due potenziometri, che permettono di regolare sia la sensibilità del sensore, ovvero la distanza massima a cui un movimento viene rilevato (solitamente dai 3 ai 7 metri), sia il tempo per cui il segnale rimane alto quando viene rilevato un movimento. Noi, durante la nostra esperienza, non abbiamo regolato questo potenziometro e abbiamo verificato che presenta un *Time Delay Adjust* di tre secondi;
2. un Trigger Selection Jumper che permette di scegliere tra due modalità operative:
    a. *Single trigger*: tempo in cui il segnale rimane HIGH a partire dal primo movimento identificato. Questo significa che il timer viene azzerato solo dopo un *Time Delay Adjust*, indipendentemente da cosa succede nel frattempo;
    b. *Repeatable trigger*: il tempo in cui il segnale rimane HIGH a partire dall'ultimo movimento rilevato.

ESERCIZIO 4:

```
void loop() {
  if(Serial.available()>0){
    int val = Serial.read();
    if (val == '+'){
      current_speed+=25.5;
      if (current_speed <= 255){
       Serial.println("Increasing speed: ");
       Serial.println(current_speed);
       analogWrite(FAN_PIN, (int) current_speed);
      }
      else{
        Serial.println("Already at max speed");
      }
    }
     else if (val == '-'){
      if (current_speed == 0){
         Serial.println("Already at min speed");
      }
      else{
       Serial.println("Decreasing speed: ");
       Serial.println(current_speed);
      current_speed-=25.5;
      analogWrite(FAN_PIN, (int) current_speed);
      }
    }
    else{
      Serial.println("Valore non riconosciuto");
    }
  }
}
```

Obiettivo controllare la velocità di rotazione di un motore a corrente continua tramite PWM.
Componenti hardware utilizzate: motore a corrente continua e porta seriale.
Componenti hardware utilizzate: segnali PWM.

Sketch

La *setup()* permette di settare i pin, di avviare una comunicazione seriale con il PC e di settare la velocità di rotazione iniziale con un duty cycle dello 0%.

La *loop()*, processando l'input della seriale, permette di incrementare o decrementare la velocità di uno step prefissato pari al 10% del totale. Se viene passato il carattere '+', la velocità aumenta di uno step, fino a raggiungere la massima velocità del motore, corrispondente a scrivere il valore 255 sul pin in output. Stessa cosa quando passiamo '-', la velocità decrementa di uno step fino ad arrivare al minimo, corrispondente a 0 sul pin. Altri valori in input sulla seriale non vengono considerati.

```
void loop() {
  // put your main code here, to run repeatedly:
  int sig = analogRead(TEMP_PIN);
  float R = ((1023.0/(float)sig) - 1.0)*R0;
  float log_sig = log(R/R0);
  float T = 1/((log_sig/B) + (1/298.15));
  float temp = T - 273.15;
  Serial.println("Temperature now: ");
  Serial.println(temp);
  delay(10000);

}
```

ESERCIZIO 5:

Obiettivo: vogliamo monitorare periodicamente la temperatura dell'ambiente circostante alla Yùn e inviare il valore al PC tramite porta seriale.

Componenti hardware utilizzate: sensore di temperatura e porta seriale

Componenti software utilizzate: input analogici

Sketch:

La *setup()* crea una comunicazione seriale con il PC e setta in input il pin a cui abbiamo collegato il nostro sensore di temperatura

La *loop()* legge il valore di tensione proveniente dal pin di input e, utilizzando le opportune formule, lo converte in un valore di temperatura, stampandolo sul Serial Monitor. Questo procedimento è ripetuto ogni 10 secondi.

Grove Temperature Sensor:

Il sensore di temperatura usato nell'esercizio produce in uscita un valore non di temperatura, ma di tensione, che a sua volta dipende dalla temperatura circostante al sensore secondo una certa relazione che dipende dalla resistenza. Per prima cosa, dobbiamo allora passare dal valore di voltaggio letto al valore di resistenza; modellizzando il sensore con due resistenze, una verso l'alimentazione Vcc, una verso massa di valore 100KOhm, otteniamo la tensione letta dal pin (quella nel nodo fra le due resistenze) con un semplice partitore della tensione Vcc, e con la relazione inversa otteniamo la resistenza che dà sull'alimentazione. Poi otteniamo il valore di temperatura servendoci della definizione della variabile Beta di un termistore (cioè resistenza non lineare di valore variabile con la temperatura), che rappresenta proprio la relazione fra il valore della sua resistenza e la temperatura. (To=298K, T è anche in kelvin)

$$Vsig = \frac{Ro}{R+Ro}Vcc \Rightarrow R = \left(\frac{Vcc}{Vsig} - 1\right)Ro \qquad B = \frac{\ln\left(\frac{R}{Ro}\right)}{\frac{1}{T}-\frac{1}{To}} \Rightarrow T = \frac{1}{\frac{\ln\left(\frac{R}{Ro}\right)}{B}+\left(\frac{1}{To}\right)}$$

ESERCIZIO 6:

```
void setup() {
  // put your setup code here, to run once:
  lcd.begin(16,2);
  lcd.setBacklight(255);
  lcd.home();
  lcd.clear();
  lcd.print("Temperature:");
  pinMode(TEMP_PIN, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int sig = analogRead(TEMP_PIN);
  float R = ((1023.0/(float)sig) - 1.0)*R0;
  float log_sig = log(R/R0);
  float T = 1/((log_sig/B) + (1/298.15));
  float temp = T - 273.15;
  lcd.home();
  lcd.clear();
  lcd.print("Temperature:" + String(temp));
  delay(10000);
}
```

Obiettivo: vogliamo visualizzare il valore di temperatura dell'esercizio precedente sul display LCD anziché trasmetterlo al PC.

Componenti hardware utilizzate: sensore di temperatura, display LCD con connessione I2C

Componenti software utilizzate: libreria LiquidCrystal_PCF8574

Sketch:

Tutte le funzionalità dell'esercizio precedente vengono riportate nel codice, meno quelle che permettono di comunicare il valore di temperatura alla porta seriale.

Oltre a queste:

La *setup()* usa delle funzioni definite nella libreria LiquidCrystal_PCF8574 per inizializzare il display:

1. *begin()*: inizializza il display a 16 colonne e 2 righe
2. *setBackLight():* imposta al massimo la retroilluminazione del display
3. *home()*:
4. *clear():* cancella qualsiasi eventuale scritta precedente sul display
5. *print()*: scriviamo "Temperature:" dall'inizio, perché sarà una scritta fissa, evitiamo di riscriverla ogni volta, in modo da minimizzare la quantità di dati inviata sul bus I2C.

La *loop()*, ottenuto il valore di temperatura corrente dal sensore sul pin di input, semplicemente lo stampa sul display con la *print()*, ripetendo il tutto a intervalli di 10 secondi.

Nel collegamento del display alla Yùn si deve solo ricordare che i pin SDA e SCL della Yùn (posti vicino ad AREF) sono collegati anche ai pin D2 e D3. È quindi necessario fare attenzione che non ci siano altri dispositivi (non I2C) connessi a tali pin, per evitare conflitti.