Versione 260124

https://github.com/Ferrazzi/Basic32

Introduzione a Basic32 – Interprete BASIC per ESP32

Basic32 è un potente ma leggero interprete BASIC sviluppato per la scheda **ESP32**, progettato per rendere la programmazione dell'ESP32 accessibile anche senza conoscenze di C/C++ o ambienti di sviluppo complessi. Con Basic32 puoi scrivere, salvare ed eseguire codice BASIC in tempo reale, utilizzando un qualsiasi terminale seriale. Questo approccio elimina completamente la necessità di ricompilare il firmware ad ogni modifica del programma.

Caratteristiche principali

- **Scrittura diretta del codice BASIC** da terminale seriale (es. Basic32Terminal, PuTTY, Arduino Serial Monitor, etc.)
- **Salvataggio e caricamento dei listati** su memoria interna (SPIFFS) o su **scheda SD** (se presente)
- **Memorizzazione del programma** in RAM con supporto a:
  - variabili numeriche, stringhe, array
  - funzioni definite dall'utente
  - flusso di controllo (IF, GOTO, GOSUB, FOR/NEXT)
- **Controllo I/O GPIO**: lettura/scrittura digitale e analogica, configurazione pin
- **Funzioni di tempo** e generazione casuale
- **Comandi integrati**: LIST, RUN, NEW, HELP, SAVE, LOAD
- **Interprete interattivo**: ogni riga può essere digitata e valutata in tempo reale

  Gestione di file BASIC tramite comandi su SD o SPIFFS

Basic32 è pensato per:

- appassionati di retro-programmazione
- maker che vogliono controllare ESP32 in modo semplice
- chi cerca un ambiente educativo e interattivo
- chi vuole fare prototipazione rapida senza compilazioni continue

Requisiti hardware

- Scheda **ESP32** (modello DEV, S3, C3 con supporto SPIFFS e interfaccia SD opzionale)
- Connessione seriale al PC
- (Opzionale) **Scheda SD** collegata ai pin definiti nel codice:
  - MOSI → GPIO 23
  - MISO → GPIO 19
  - SCK → GPIO 18
  - CS → GPIO 15

Utilizzo di più dispositivi SPI su ESP32 con Basic32

L'ESP32 dispone di bus SPI hardware che possono essere condivisi tra più periferiche (display TFT, lettore RFID, touch screen, scheda SD, ecc.). Tutti i dispositivi SPI utilizzano

in comune i pin **MOSI**, **MISO** e **SCK**, ma ciascun dispositivo deve avere un **Chip Select (CS)** dedicato.

Per evitare conflitti sul bus SPI:

1. **Assegna un pin CS dedicato a ogni dispositivo** (es. TFT_CS=17, SD_CS=13, TOUCH_CS=4, RFID_CS=25).
2. **Mantieni i CS alti (HIGH)** quando il dispositivo non è in uso, così non interferisce con gli altri.
3. **Imposta i CS come OUTPUT e portali HIGH prima di inizializzare i moduli**.
4. **Inizializza i dispositivi** con i relativi comandi (INITSD, ILI INIT, RFID INIT, ecc.).

Esempio: Display ILI9341 con SD e Touch integrati

```
10 PINMODE 17 OUTPUT NOPULL   ' TFT_CS come output
20 DWRITE 17 1                ' tiene inattivo il display
30 PINMODE 4 OUTPUT NOPULL    ' TOUCH_CS come output
40 DWRITE 4 1                 ' tiene inattivo il touch
50 PINMODE 13 OUTPUT NOPULL   ' SD_CS come output
60 DWRITE 13 1                ' tiene inattiva sd
70 INITSD 13 23 19 18         ' inizializza SD (CS=13, MOSI=23, MISO=19, SCK=18)
80 ILI INIT 17 16 5 3         ' inizializza TFT (CS=17, DC=16, RST=5, rotazione=3)
90 ILI LED 32 1               ' accende retroilluminazione su GPIO32
100 ILI TEXT 10 50 2 SDFREE 0 255 255  ' stampa spazio libero su SD in giallo
```

Nota: Se aggiungi altre periferiche SPI (es. RFID RC522), assegna loro un CS dedicato, imposta PINMODE <CS> OUTPUT e DWRITE <CS> 1 prima di eseguire il rispettivo INIT.

Con questa sequenza, ogni dispositivo è pronto a lavorare senza disturbare gli altri sul bus SPI.

Cosa puoi fare con Basic32?

- Scrivere e testare **algoritmi BASIC** in tempo reale
- Costruire **applicazioni interattive** su ESP32 senza compilare
- **Salvare programmi** per riutilizzarli o modificarli in futuro
- Controllare sensori e attuatori con semplici comandi BASIC

Installazione e Primo Avvio

Questa sezione ti guida passo passo nell'installazione di **Basic32** su una scheda **ESP32**, utilizzando firmware già compilati.
Non è necessario usare l'Arduino IDE: il firmware può essere flashato **direttamente dal browser** oppure tramite **script di installazione** forniti dal progetto.

---

1. Requisiti

- Scheda **ESP32** compatibile:
  - ESP32 DEV
  - ESP32 S3
  - ESP32 C3
- Cavo USB per collegare l'ESP32 al PC
- PC con **browser compatibile WebSerial** (Chrome, Edge o derivati)
  oppure
- Ambiente con **Python** installato (per installazione tramite script)

---

2. Metodi di Installazione

Metodo 1 – Installazione diretta dal sito (consigliato)

Il modo più semplice e veloce per installare Basic32 è tramite il sito ufficiale, che permette di flashare il firmware direttamente sull'ESP32 dal browser.

1. Collega la scheda ESP32 al PC tramite USB
2. Apri il sito:
3. https://ferrazzi.github.io/Basic32/
4. Clicca su **Connect**
5. Autorizza l'accesso alla porta seriale quando richiesto
6. Attendi il completamento del flash

Al termine, la scheda è pronta per l'uso con Basic32.

Metodo 2 – Installazione tramite firmware e script (avanzato)

In alternativa, è possibile installare Basic32 scaricando i file del firmware e utilizzando gli script ufficiali forniti nel repository GitHub del progetto.

1. Vai alla sezione firmware del progetto:
2. https://github.com/Ferrazzi/Basic32/tree/main/firmware
3. Scarica la cartella relativa al tuo modello di ESP32
4. Collega l'ESP32 al PC
5. Esegui lo script di installazione fornito (Windows, Linux o macOS)

6. Attendi il completamento del flash

Questo metodo è indicato per utenti più esperti o per installazioni offline.

---

3. Primo Avvio

1. Una volta flashato, riavvia l’ESP32.
2. Apri un terminale seriale a **115200 baud**.
3. Dovresti vedere il prompt:

   ```
   BASIC32 v1.0 READY
   ```

   Ora puoi digitare comandi BASIC direttamente:

   ```
   10 PRINT "HELLO BASIC32"
   20 GOTO 10
   RUN
   ```

---

4. Utilizzo SPIFFS

...puoi salvare e caricare listati BASIC con i comandi:

```
SAVE "programma.bas"
LOAD "programma.bas"
```

---

5. Utilizzo scheda SD (opzionale)

Se il tuo hardware ha una scheda SD collegata ai seguenti pin:

Segnale	GPIO ESP32
MISO	19
MOSI	23
SCK	18
CS	15

Gestione File e Memoria

Basic32 supporta sia **la memoria interna SPIFFS** dell'ESP32, sia **una scheda microSD** opzionale. Entrambi i supporti possono essere utilizzati per **salvare, caricare e organizzare i file BASIC** (.bas) senza dover ricompilare il firmware.

---

1. Memoria SPIFFS (interna)

SPIFFS è il file system interno dell'ESP32, montato automaticamente all'avvio. È utile quando non si ha a disposizione una scheda SD.

Note:

- I nomi dei file sono **case-insensitive**.
- L'estensione .bas è convenzionale, ma non obbligatoria.
- La dimensione disponibile dipende dalla partizione SPIFFS nel firmware (tipicamente 1MB–2MB).

---

2. Scheda SD (esterna, opzionale)

Se hai una scheda microSD collegata all'ESP32 (con pin configurati nel file Basic32.ino), puoi utilizzarla come **memoria aggiuntiva** o principale.

- Il sistema **rileva automaticamente la presenza** della scheda SD.
- La SD deve essere formattata in FAT32

La SD deve essere formattata in** FAT32**.

ABS(x)

Sintassi:

ABS(x)

Descrizione:
La funzione ABS(x) restituisce il **valore assoluto** di x, cioè il numero **senza segno**.
È utilizzabile in espressioni aritmetiche, assegnazioni e condizioni logiche.

Accetta sia numeri interi che decimali. Se il numero è già positivo o zero, non viene modificato.

Esempi pratici

Esempio 1 – Valore assoluto di un intero negativo
→ Mostra l'uso di ABS con un numero intero:

```
10 A = -42
20 B = ABS(A)
30 PRINT "VALORE ASSOLUTO: "; B
RUN
```

Output atteso:

```
VALORE ASSOLUTO: 42
```

Esempio 2 – Valore assoluto con numero decimale
→ Funziona anche con numeri float (virgola mobile):

```
10 PRINT "ABS(-3.14) = "; ABS(-3.14)
RUN
```

Output atteso:

```
ABS(-3.14) = 3.14
```

Esempio 3 – Uso diretto in condizione
→ ABS può essere usato direttamente in una condizione IF:

```
10 A = -7
20 IF ABS(A) = 7 THEN PRINT "È UGUALE A 7"
RUN
```

Output atteso: È UGUALE A 7

ACS CALIB SETOFFSET

Sintassi

ACS CALIB SETOFFSET mv

Descrizione

Imposta manualmente l'**offset di zero** in **mV** (tipicamente ~Vcc/2). Utile se conosci l'offset misurato con multimetro o vuoi forzarlo.

Esempio

```
10 ACS INIT 34 20
20 ACS CALIB SETOFFSET 1650
30 ACS READ I
40 PRINT I
```

Note

- Non sostituisce la calibrazione automatica; puoi combinare **ZERO** e **SETOFFSET** (l'ultimo chiamato vince).

ACS CALIB SHOW

Sintassi

ACS CALIB SHOW

Descrizione
Mostra su seriale la configurazione attuale: pin, modello, sensibilità mV/A, vref, zero in mV, campioni mediati.

Esempio

```
10 ACS INIT 34 5
20 ACS CALIB SHOW
```

Note

- Output via Serial.printf(...) (come gli altri comandi di diagnostica che stampano su seriale).

ACS CALIB ZERO

Sintassi

ACS CALIB ZERO [samples]

Descrizione

Esegue la **calibrazione dello zero** (nessuna corrente nel sensore!). Legge samples campioni e fissa l'offset in mV.

Esempio – Calibrazione accurata

```
10 ACS INIT 34 5
20 PRINT "Togli corrente e premi INVIO"
30 WAIT 3000
40 ACS CALIB ZERO 256
50 ACS CALIB SHOW
```

Note

- Assicurati che **non scorra corrente** nel cavo durante la calibrazione.
- Più campioni → offset più stabile (consiglio 128–512).

ACS INIT (acs712)

Sintassi

ACS INIT pin model [vref_mv] [avgSamples]

Descrizione

Inizializza il sensore ACS712.

- pin: GPIO **ADC1** consigliato (32–39).
- model: **5**, **20** o **30** (Ampere) → imposta automaticamente mV/A (185 / 100 / 66).
- vref_mv: opzionale (fallback se non si usa analogReadMilliVolts).
- avgSamples: opzionale; default 32.

Esempio – Modulo 5 A su GPIO34

10 ACS INIT 34 5
20 ACS CALIB SHOW

Note (HW)

- Collega **OUT** del modulo ACS712 al pin ADC scelto (meglio **ADC1**).
- Alimentazione del modulo secondo specifiche (spesso 5 V). L'uscita è **centrata a Vcc/2**.
- Per ESP32: attenuazione 11 dB (~3.3 V full-scale) già impostata nel codice.

ACS READ

Sintassi

ACS READ var

Descrizione

Legge la **corrente DC media** (in **Ampere**) usando avgSamples campioni. Salva il valore in var.

Esempio – Lettura continua

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 ACS READ I
40 PRINT "I=";I;" A"
50 WAIT 200
60 GOTO 30
```

Note

- Per **DC** o AC rettificata/filtrata. Per **AC pura**, usa **ACS RMS**.

ACS RMS

Sintassi

ACS RMS window_ms var

Descrizione

Misura la **corrente AC RMS** (Ampere) su una finestra temporale di window_ms.
Rimuove l'offset (zero) ad ogni campione e calcola sqrt(media(x^2)).

Esempio – 50 Hz su 500 ms

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 ACS RMS 500 IRMS
40 PRINT "Irms=";IRMS;" A"
50 WAIT 500
60 GOTO 30
```

Note

- Per 50 Hz, finestre di **200–500 ms** sono ok. Più lunga = più stabile.
- Assicurati di aver calibrato lo zero **senza carico**.

ACS SAMPLES

Sintassi

ACS SAMPLES n

Descrizione
Imposta il numero di **campioni** per le letture **DC** mediate (ACS READ).

Esempio

```
10 ACS INIT 34 5
20 ACS SAMPLES 64
30 ACS READ I
40 PRINT I
```

Note

- Non influisce su **ACS RMS**, che usa la sua finestra temporale.

ACS SENS

Sintassi

ACS SENS mv_per_A

Descrizione
Imposta manualmente la **sensibilità** (mV/A). Utile se il tuo modulo non è il classico ACS712 o vuoi tarare la pendenza.

Esempio – Forza 100 mV/A

```
10 ACS INIT 34 5
20 ACS SENS 100
30 ACS READ I
40 PRINT I
```

Note

- I valori tipici ACS712: 185 (5 A), 100 (20 A), 66 (30 A).

ACS VREF

Sintassi

ACS VREF mv

Descrizione
Imposta la **tensione di riferimento** ADC in mV (usata solo come **fallback** se non è disponibile analogReadMilliVolts).

Esempio

```
10 ACS INIT 34 20 3300
20 ACS VREF 3300
```

Note

- Su ESP32 con analogReadMilliVolts **non serve**. Lascia 3300 mV.

ACS WATCH

Sintassi

ACS WATCH limit GOTO line
ACS WATCH limit ABOVE GOTO line
ACS WATCH limit BELOW GOTO line
ACS WATCH limit GOTO line RMS window_ms
ACS WATCH limit ABOVE|BELOW GOTO line RMS window_ms

Descrizione

Esegue una lettura di corrente e **salta** alla riga line se la condizione è vera.

- Default: **ABOVE** (salta se corrente ≥ limit).
- Con **BELOW**: salta se corrente ≤ limit.
- Con **RMS window_ms** misura **AC RMS** sulla finestra specificata; senza RMS usa la **media DC**.

Esempi pratici

Esempio 1 – Protezione sovracorrente (DC)

```
10 ACS INIT 34 20
20 ACS CALIB ZERO 256
30 PRINT "Loop motore..."
40 ' ... comandi per il motore qui ...
50 ACS WATCH 2.5 GOTO 200
60 WAIT 50
70 GOTO 40
200 PRINT "ALLARME: sovracorrente!"
210 ' Spegni/ferma qui il carico
```

Esempio 2 – Soglia minima (caduta corrente)

```
10 ACS INIT 34 5
20 ACS CALIB ZERO 256
30 PRINT "Monitor corrente minima"
40 ACS WATCH 0.20 BELOW GOTO 100
50 WAIT 200
60 GOTO 40
100 PRINT "Corrente bassa! Verifica carico/cavo."
```

Esempio 3 – AC a 50 Hz con RMS (finestra 400 ms)

```
10 ACS INIT 34 30
20 ACS CALIB ZERO 256
30 PRINT "Controllo AC RMS"
40 ACS WATCH 0.80 GOTO 100 RMS 400
50 WAIT 100
60 GOTO 40
100 PRINT "Soglia Irms raggiunta!"
```

Esempio 4 – Stop servo al supero soglia

```
10 ACS INIT 34 5
```

```
20 ACS CALIB ZERO 256
30 PRINT "Servo in movimento"
40 ' SERVOWRITE 1 90   ' esempio
50 ACS WATCH 1.2 GOTO 300
60 WAIT 50
70 GOTO 40
300 PRINT "STOP: corrente troppo alta"
310 ' SERVOWRITE 1 0
```

Note

- Usa **in loop**; il comando non mantiene stato interno: valuta e, se vero, **salta**.
- Con **RMS**, la chiamata è **bloccante** per ~window_ms. Scegli una finestra adeguata (per 50 Hz tipicamente 200–500 ms) o resta su DC se vuoi risposte più rapide.
- Assicurati di aver fatto **ACS CALIB ZERO senza carico** per uno zero stabile.
- Puoi combinare più ACS WATCH nel loop con soglie diverse (es. warning e shutdown).

ADC CAL GAIN <fattore>

Sintassi:

ADC CAL GAIN <fattore>

Descrizione:

Imposta il **guadagno moltiplicativo** del convertitore ADC.
Serve per correggere differenze di scala tra la tensione reale e quella misurata.
Il valore predefinito è 1.0.

Esempi pratici

Esempio 1 – Aumentare la scala dell'1.5 %

ADC CAL GAIN 1.015

⟶ Moltiplica tutte le letture per 1.015 (cioè +1.5%).

Esempio 2 – Ridurre la scala del 2 %

ADC CAL GAIN 0.98

⟶ Riduce tutte le letture del 2%.

Output atteso

ADC REF=3.300000 GAIN=1.015000 OFFSET=0.000000

Nota:

• GAIN deve essere > 0, altrimenti errore:

?ADC CAL ERROR
GAIN must be > 0

• Tipicamente varia tra 0.95 e 1.10.
• Si applica automaticamente a VREAD e RREAD.

ADC CAL MEASURE <pin> <volt_noto> [campioni]

Sintassi:

ADC CAL MEASURE <pin> <volt_noto> [campioni]

Descrizione:

Esegue una **calibrazione automatica** del GAIN misurando una tensione nota presente sul pin ADC.
Il firmware legge la tensione reale al pin (non calibrata), la confronta con il valore noto e aggiorna il GAIN in modo che le future letture coincidano.

Esempi pratici

Esempio 1 – Calibrazione a 3.300 V su GPIO34

ADC CAL RESET
ADC CAL REF 3.30
ADC CAL MEASURE 34 3.300 32

⟶ Esegue 32 campioni sul pin 34, calcola la media e regola il GAIN in modo che la lettura risulti 3.300 V.

Esempio 2 – Calibrazione a 2.500 V con 16 campioni

ADC CAL MEASURE 34 2.500 16

⟶ Calibra il GAIN leggendo 16 volte la tensione nota di 2.500 V.

Output atteso

ADC CAL GAIN computed = 1.160772

Verifica poi con:

ADC CAL STATUS

Output:

ADC REF=3.300000 GAIN=1.160772 OFFSET=0.000000

Nota:

• volt_noto deve corrispondere a una **tensione reale e stabile** sul pin.
• [campioni] è facoltativo (default 16).
• Valori raccomandati: tra 1.0 V e 3.0 V.
• Se la tensione è 0 o assente:

?ADC CAL ERROR
Measured ~0V (check wiring/ref)

• MEASURE aggiorna solo GAIN (non REF o OFFSET).

ADC CAL OFFSET <volt>

Sintassi:

ADC CAL OFFSET <volt>

Descrizione:

Applica un **offset additivo** (in volt) a tutte le letture ADC.
Serve per correggere errori di zero o differenze costanti tra 0 V reale e lettura ADC.

Esempi pratici

Esempio 1 – Sottrarre 10 mV

ADC CAL OFFSET -0.010

⟶ Corregge una piccola sovrastima costante delle misure.

Esempio 2 – Aggiungere 5 mV

ADC CAL OFFSET 0.005

⟶ Compensa una sottostima delle letture.

Output atteso

ADC REF=3.300000 GAIN=1.000000 OFFSET=-0.010000

Nota:

• OFFSET si somma *dopo* il guadagno (V = (Raw × REF / Counts) × GAIN + OFFSET).
• Valori tipici: da −0.020 V a +0.020 V.
• Utile per correggere offset costanti a 0 V.
• Nessun errore se impostato a 0.

ADC CAL REF <volt>

Sintassi:

ADC CAL REF <volt>

Descrizione:

Imposta la **tensione di riferimento ADC**, cioè il valore massimo corrispondente alla piena scala della conversione.
Serve per adattare il calcolo dei volt all'alimentazione reale del microcontrollore.

Esempi pratici

Esempio 1 – Impostare 3.30 V

ADC CAL REF 3.30

⟶ Usa 3.30 V come riferimento standard (ESP32).

Esempio 2 – Riferimento personalizzato

ADC CAL REF 3.28

⟶ Imposta il riferimento alla tensione realmente misurata di 3.28 V.

Output atteso

ADC REF=3.280000 GAIN=1.000000 OFFSET=0.000000

Nota:

• REF deve essere > 0, altrimenti errore:

?ADC CAL ERROR
REF must be > 0

• Impostalo sul valore reale della tensione di alimentazione (es. 3.28 V).
• Si combina con GAIN e OFFSET per calcolare la tensione finale.

ADC CAL RESET

Sintassi:

ADC CAL RESET

Descrizione:

Ripristina i valori di calibrazione ai **default di fabbrica**:

- REF = 3.30 V
- GAIN = 1.0
- OFFSET = 0.0

Utile per annullare qualsiasi calibrazione precedente e tornare ai parametri standard.

Esempi pratici

Esempio 1 – Reset completo

ADC CAL RESET
ADC CAL STATUS

⟶ Riporta tutti i parametri ai valori originali.

Output atteso

ADC REF=3.300000 GAIN=1.000000 OFFSET=0.000000

Nota:

• Nessun errore se eseguito più volte.
• Eseguire prima di una nuova calibrazione (ADC CAL MEASURE).
• Garantisce letture consistenti e ripetibili.

ADC CAL STATUS

Sintassi:

ADC CAL STATUS

Descrizione:

Mostra i **parametri di calibrazione** attualmente attivi:

- Riferimento (REF)
- Guadagno (GAIN)
- Offset (OFFSET)

Permette di verificare i valori applicati a VREAD e RREAD.

Esempio pratico

ADC CAL STATUS

⟶ Stampa i parametri correnti sul monitor seriale.

Output atteso

ADC REF=3.300000 GAIN=1.160772 OFFSET=0.000000

Nota:

• Non modifica alcun valore.
• Usalo dopo ogni calibrazione o variazione per confermare i parametri.
• È il comando di verifica principale del sistema ADC.

ALEXABRIGHT

Sintassi

ALEXABRIGHT nome

Descrizione

Il comando **ALEXABRIGHT** restituisce il valore di luminosità (dimmer) inviato da Alexa per un dispositivo associato.

Il valore restituito è compreso tra **0 e 255**, dove:

0 → dispositivo spento (OFF)

255 → luminosità massima (100%)

valori intermedi → luminosità proporzionale

Il comando:

NON consuma l'evento

può essere utilizzato in **PRINT, IF, LET**

è ideale per LED PWM, luci dimmerabili e controlli analogici

Esempi pratici

Esempio 1 – Visualizzare il valore di luminosità Alexa

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"

40 PRINT "Luminosità:"; ALEXABRIGHT "Luce"
RUN
```

Output atteso:
Stampa un valore da 0 a 255 in base al comando Alexa.

Esempio 2 – Controllo LED PWM con Alexa

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20

30 PINMODE 25 OUTPUT NOPULL
40 ALEXACALL "Luce"

60 LET B = ALEXABRIGHT "Luce"
70 AWRITE 25 B
80 GOTO 60
RUN
```

Output atteso:
La luminosità del LED sul pin 25 segue il livello impostato da Alexa.

Esempio 3 – Visualizzare la luminosità in percentuale

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"

40 LET B = ALEXABRIGHT "Luce"
50 LET P = INT(B * 100 / 255)
60 PRINT "Luminosità:"; P; "%"
70 GOTO 40
RUN
```

Note

• Il valore è sempre compreso tra **0 e 255**
• OFF equivale a **0**

• Il comando **legge** il valore, non lo modifica
• Può essere combinato con **ALEXASTATE**

ALEXACALL

Sintassi

ALEXACALL nome GOTO linea

Descrizione

Il comando **ALEXACALL** registra un dispositivo Alexa e consente di intercettare i comandi vocali come **eventi**.

Funzioni principali:

crea un dispositivo Alexa

intercetta comandi ON/OFF

può forzare un **salto immediato (GOTO)** nel programma

Il comando è **one-shot**:

restituisce **1 una sola volta** quando arriva un evento

dopo la lettura l'evento viene consumato

Ideale per:

pulsanti

cancelli

serrature

attuatori momentanei

Esempi pratici

Esempio 1 – Pulsante Alexa con GOTO

```
10 WIFI "ssid" "password"

20 IF WIFICONNECTED = 0 THEN GOTO 20


30 ALEXACALL "Apri Cancello" GOTO 100


40 PRINT "In attesa comando..."
```

```
50 DELAY 1000
60 GOTO 40

100 PRINT "COMANDO RICEVUTO"
110 GOTO 40
RUN
```

Esempio 2 – Evento Alexa con IF

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Pulsante"

40 IF ALEXACALL "Pulsante" THEN PRINT "PREMUTO!"
50 DELAY 500
60 GOTO 40
RUN
```

Esempio 3 – Attuatore momentaneo

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20

30 PINMODE 2 OUTPUT NOPULL
40 ALEXACALL "Impulso" GOTO 100

60 DELAY 1000
70 GOTO 60

100 DWRITE 2 1
```

110 DELAY 300

120 DWRITE 2 0

130 GOTO 60

RUN

Note

• L’evento viene **consumato** dopo la lettura
• Ideale come **pulsante**
• Per ON/OFF continuo usare **ALEXASTATE**
• Richiede Wi-Fi attivo

ALEXASTATE

Sintassi

ALEXASTATE nome

Descrizione

Il comando **ALEXASTATE** restituisce lo **stato attuale ON/OFF** di un dispositivo Alexa.

Valori restituiti:

1 → dispositivo acceso

0 → dispositivo spento

Il comando:

NON consuma l'evento

può essere letto più volte

è ideale per pilotare LED, relè e carichi reali

Esempi pratici

Esempio 1 – Stampare lo stato Alexa

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce"
40 PRINT ALEXASTATE "Luce"
RUN
```

Esempio 2 – Accendere e spegnere un LED

```
10 WIFI "ssid" "password"
20 IF WIFICONNECTED = 0 THEN GOTO 20
30 ALEXACALL "Luce" GOTO 60
40 DELAY 200
```

```
50 GOTO 40

60 IF ALEXASTATE "Luce" = 1 THEN PRINT "ON"

70 IF ALEXASTATE "Luce" = 0 THEN PRINT "OFF"

80 GOTO 40

RUN
```

Esempio 3 – Stato + evento combinati

```
10 WIFI "ssid" "password"

20 IF WIFICONNECTED = 0 THEN GOTO 20

30 ALEXACALL "Luce"


40 IF ALEXACALL "Luce" THEN PRINT "EVENTO"

50 PRINT "STATO:"; ALEXASTATE "Luce"

60 DELAY 500

70 GOTO 40

RUN
```

Note

- OFF equivale a **0**
- Lo stato rimane valido finché Alexa non lo modifica
- Può essere usato in **IF, LET, PRINT**
- Ideale per relè e luci reali

AREAD(p)

Sintassi:

AREAD(p)

Descrizione:
La funzione AREAD(p) legge il valore **analogico** dal pin p dell'ESP32.
Restituisce un valore compreso tra **0 e 4095**, dove:

- 0 corrisponde a **0V**
- 4095 corrisponde a circa **3.3V**

È usato per leggere sensori analogici (es. potenziometri, sensori di luce, temperatura, ecc.) collegati a uno dei **pin analogici dell'ESP32**.
Pin comuni per la lettura analogica includono: **GPIO 36, 39, 34, 35, 32, 33**.

 I pin **digitali normali** non supportano lettura analogica. Assicurati di usare i pin ADC corretti.

Esempi pratici

Esempio 1 – Lettura continua da un potenziometro
→ Legge un valore dal pin GPIO36 ogni secondo:

```
10 PRINT "LETTURA ANALOGICA:"
20 V = AREAD(36)
30 PRINT "VALORE: "; V
40 WAIT 1000
50 GOTO 20
RUN
```

Output atteso:
(valori variabili da 0 a 4095, dipende dalla posizione del potenziometro)

```
LETTURA ANALOGICA:
VALORE: 512
...
```

Esempio 2 – Verifica soglia di luminosità
→ Accende un LED se la luce scende sotto una soglia (simulazione):

```
10 LUX = AREAD(36)
20 IF LUX < 1000 THEN PRINT "LUCE OK" ELSE PRINT "LUCE BASSA"
30 WAIT 1000
40 GOTO 10
RUN
```

Output atteso:

LUCE OK
LUCE BASSA

AND, OR, NOT (Operatori Logici)

Sintassi:

A AND B
A OR B
NOT A

Descrizione:
Gli operatori logici AND, OR e NOT sono usati per eseguire confronti **logici o bit a bit** all'interno di espressioni condizionali o aritmetiche.

- AND restituisce 1 solo se **entrambi** gli operandi sono diversi da zero
- OR restituisce 1 se **almeno uno** dei due è diverso da zero
- NOT inverte il valore logico: NOT 0 è -1, NOT 1 è 0

In BASIC32, questi operatori possono essere usati:

- Nei confronti logici (IF ... THEN)
- In assegnazioni (LET)
- In operazioni binarie (es. maschere di bit)

Esempi pratici

Esempio 1 – Uso con IF e AND

```
10 A = 1
20 B = 2
30 IF A = 1 AND B = 2 THEN PRINT "ENTRAMBI VERI"
RUN
```

Output atteso:

```
ENTRAMBI VERI
```

Esempio 2 – Uso con OR

```
10 A = 0
20 B = 5
30 IF A <> 0 OR B <> 0 THEN PRINT "ALMENO UNO È DIVERSO DA ZERO"
RUN
```

Output atteso:

```
ALMENO UNO È DIVERSO DA ZERO
```

Esempio 3 – Uso di NOT

```
10 A = 0
20 IF NOT A THEN PRINT "A È ZERO"
RUN
```

Output atteso:

```
A È ZERO
```

Esempio 4 – Maschera di bit con AND

```
10 X = 7        ' binario: 0111
20 MASK = 4     ' binario: 0100
30 RESULT = X AND MASK
40 PRINT "RISULTATO: "; RESULT
RUN
```

Output atteso:

```
RISULTATO: 4
```

Esempio 5 – Uso in assegnazione logica

```
10 A = 5
20 B = (A > 0) AND (A < 10)
30 PRINT B
RUN
```

Output atteso:

```
1
```

Nota:

- AND, OR, NOT restituiscono valori numerici (0 o 1/-1)
- Valori diversi da zero sono trattati come **VERO** (TRUE)
- Valori uguali a zero sono **FALSO** (FALSE)

ASC(A$)

Sintassi:

ASC(stringa$)

Descrizione:
La funzione ASC restituisce il **codice ASCII** del **primo carattere** della stringa A$.
È utile per:

- Identificare il valore numerico di un carattere
- Creare confronti tra lettere
- Gestire input tastiera carattere per carattere (es. con GET)

Se la stringa è vuota (""), il risultato è **0** oppure può generare un errore (a seconda dell'implementazione).

Esempi pratici

Esempio 1 – Codice ASCII di una lettera

```
10 A$ = "A"
20 PRINT ASC(A$)
RUN
```

Output atteso:

```
65
```

Esempio 2 – Confronto con una lettera specifica

```
10 C$ = "Z"
20 IF ASC(C$) = 90 THEN PRINT "È Z"
RUN
```

Output atteso:

```
È Z
```

Esempio 3 – Da carattere a codice e ritorno

```
10 T$ = "C"
20 COD = ASC(T$)
30 PRINT CHR$(COD)
RUN
```

Output atteso:

C

Esempio 4 – Lettura multipla da stringa

```
10 S$ = "ABC"
20 FOR I = 1 TO LEN(S$)
30 PRINT MID$(S$, I, 1); " = "; ASC(MID$(S$, I, 1))
40 NEXT I
RUN
```

Output atteso:

```
A = 65
B = 66
C = 67
```

Nota:

- Solo il **primo carattere** della stringa è considerato
- Se la stringa è vuota (""), può restituire 0 o generare errore
- Usare insieme a CHR$, LEFT$, GET, MID$ per manipolazioni complesse

AUTORUN

Sintassi:

AUTORUN "file.bas"
AUTORUN "file.bas" PIN <numero>
AUTORUN OFF

Descrizione:

Il comando AUTORUN imposta l'esecuzione automatica di un programma BASIC all'avvio del sistema. Il nome del file da eseguire deve essere racchiuso tra virgolette e deve avere estensione .bas.

È possibile specificare un PIN di sicurezza (PIN <numero>), ovvero un GPIO che può essere letto all'avvio per decidere se eseguire o meno l'autorun (implementazione opzionale lato firmware).

Se viene usato AUTORUN OFF, l'autorun viene disattivato e il file di configurazione viene rimosso.

Se non viene specificato alcun PIN, viene automaticamente usato PIN 0.

Esempi pratici

Esempio 1 – Abilitare autorun su un file

AUTORUN "startup.bas"

⟶ Avvierà automaticamente startup.bas all'accensione, con PIN 0 come predefinito.

Esempio 2 – Abilitare autorun con un PIN specifico

AUTORUN "demo.bas" PIN 5

⟶ Salva la configurazione per eseguire demo.bas all'avvio, con controllo su GPIO5.

Esempio 3 – Disattivare l'autorun

AUTORUN OFF

⟶ Disattiva completamente l'avvio automatico.

Output atteso:

AUTORUN active on: startup.bas
Safe PIN has been configured: GPIO0

Oppure in caso di disattivazione:

AUTORUN disattivato.

Nota:

- Il file specificato deve esistere su SPIFFS ed avere estensione .bas
- Se il file non viene trovato, viene restituito un errore
- Il GPIO di sicurezza è opzionale e può essere usato per bloccare l’esecuzione automatica a seconda del valore logico del GPIO
- Il file di configurazione /autorun.cfg viene sovrascritto ogni volta che si imposta un nuovo autorun
- L'esecuzione del programma avviene **immediatamente** dopo la configurazione

AWRITE(p, v)

Sintassi:

AWRITE(pin valore)

Descrizione:

Il comando AWRITE imposta un **segnale PWM** (modulazione di larghezza d'impulso) su un pin dell'ESP32, simulando un valore analogico.

È utilizzato per:

- **Regolare la luminosità di un LED**
- **Controllare la velocità di un motore**
- **Gestire dispositivi analogici via PWM**

Il **valore** deve essere compreso tra 0 e 255, dove:

- 0 → segnale completamente spento (0% duty cycle)
- 255 → segnale al massimo (100% duty cycle)
- valori intermedi → proporzionali (es. 128 = 50%)

Il pin deve essere prima configurato come OUTPUT usando PINMODE.

Esempi pratici

Esempio 1 – Luminosità LED al 50%

```
10 PINMODE 13 OUTPUT NOPULL
20 AWRITE 13 128
RUN
```

Output atteso: Il LED collegato al GPIO13 si accende a metà intensità.

Esempio 2 – Fading continuo (effetto dissolvenza)

```
10 PINMODE 2 OUTPUT NOPULL
20 FOR A = 0 TO 255 STEP 5
30 AWRITE 2 A
40 DELAY 50
50 NEXT A
60 FOR I = 255 TO 0 STEP -5
70 AWRITE 2 I
80 DELAY 50
90 NEXT I
```

```
RUN
```

Output atteso: Il LED collegato al GPIO12 aumenta e poi diminuisce gradualmente la luminosità.

Esempio 3 – Controllo analogico basato su variabile

```
10 PINMODE 14 OUTPUT NOPULL
20 INPUT L
30 AWRITE 14 L
RUN
```

Output atteso: Il valore della variabile `L` (es. da un sensore) regola la luminosità del LED sul pin 14.

Note:

- `AWRITE` richiede ESP32 con **Arduino core 3.x o superiore**, dove `analogWrite()` è disponibile.
- Se il pin non supporta PWM, il comando potrebbe non avere effetto visibile.
- I valori oltre `255` vengono automaticamente limitati a `255`.

BREAKPIN

Sintassi:

BREAKPIN pin
BREAKPIN pin PULLUP|PULLDOWN|NOPULL
BREAKPIN OFF
BREAKPIN ?

Descrizione:
Imposta un “pin di sicurezza” che, se premuto, interrompe immediatamente il programma BASIC in esecuzione.
• pin (obbligatorio): GPIO da usare come pulsante di stop.
• Modalità di ingresso:
– PULLUP (default): il pin usa la pull-up interna; il pulsante va a GND; premuto = LOW.
– PULLDOWN: usa la pull-down interna; il pulsante va a VCC; premuto = HIGH.
– NOPULL: nessuna resistenza interna (usa hardware esterno); premuto = HIGH.
• OFF: disabilita la funzione.
La configurazione viene salvata in EEPROM e ricaricata all’avvio.

Esempi pratici
Stop con pull-up (pulsante verso GND)
BREAKPIN 0

Stop con pull-down (pulsante verso VCC)
BREAKPIN 12 PULLDOWN

Senza pull interni (hardware esterno)
BREAKPIN 25 NOPULL

Disabilitare il BREAKPIN
BREAKPIN OFF

Note:
• L’effetto è immediato: alla pressione il programma si arresta senza attendere la fine della riga corrente (il controllo avviene all’inizio di ogni riga eseguita).
• Evita pin critici di boot (es. 0, 2, 15) se il tuo hardware non li gestisce correttamente all’avvio.
• Con PULLUP il pulsante deve chiudere a massa; con PULLDOWN a VCC.
• Se il pin non è configurato (o è OFF), nessun arresto avviene.
• La funzione è indipendente dal SAFE MODE di autorun: SAFE MODE blocca l’avvio automatico, BREAKPIN ferma l’esecuzione mentre il programma gira.

BT AT

Sintassi

```
BT AT "cmd" RESP var$ [timeout]
```

Descrizione

Invia un comando AT (aggiunge automaticamente CRLF) e legge **una riga di risposta** entro timeout millisecondi.
Se timeout è omesso, usa il valore impostato con BT TIMEOUT.
La risposta viene scritta in var$.

Esempi

Esempio 1 – Test AT

```
10 BT ATMODE ON
20 BT AT "AT" RESP R$ 1000
30 PRINT R$
```

Output atteso:

```
OK
```

Esempio 2 – Leggere il nome del modulo

```
10 BT ATMODE ON
20 BT AT "AT+NAME?" RESP N$ 1000
30 PRINT N$
```

Output atteso:

```
+NAME:BASIC32
```

Esempio 3 – Diagnostica baud AT

```
10 BT INIT 15 12 9600 33
20 BT ATMODE OFF
30 BT AT "AT" RESP R$ 500
40 PRINT "DATA9600=[";R$;"]"
50 BT END
60 BT INIT 15 12 38400 33
70 BT ATMODE OFF
```

```
80 BT AT "AT" RESP R$ 500
90 PRINT "DATA38400=[";R$;"]"
```

Output tipico:

```
DATA9600=[]
DATA38400=[OK]
```

Note

Richiede AT mode attivo (BT ATMODE ON) o KEY alto all’avvio (HC-05).

Su molti HC-05 il vero AT è a **38400**.

BT ATMODE OFF

Sintassi:
BT ATMODE OFF

Descrizione:
Esce dall'AT mode: porta KEY basso (se configurato) e riapre la UART alla **baud dati** salvata.

Esempi pratici

```
10 BT ATMODE OFF
20 PRINT "DATA MODE"
RUN
```

Output atteso:

```
DATA MODE
```

Nota:
• Dopo avere cambiato baud in AT, riallinea poi la porta dati con BT INIT

BT ATMODE ON

Sintassi:
BT ATMODE ON [keyPin]

Descrizione:
Entra in **AT mode**: porta KEY alto (usa keyPin se passato, altrimenti quello di BT INIT) e riapre la UART all'AT baud (tipicamente **38400** su HC-05).

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 BT ATMODE ON
30 BT AT "AT" RESP R$ 1000
40 PRINT R$
RUN
```

Output atteso:

```
OK
```

Nota:
• Su molti HC-05 il vero AT richiede KEY alto **all'accensione**
• Alcuni HC-06 restano in AT a 9600

BT AVAILABLE

Sintassi:
BT AVAILABLE var

Descrizione:
Scrive in var quanti **byte** sono disponibili nel buffer RX della UART BT.

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 BT AVAILABLE N
30 PRINT N
RUN
```

Output atteso:

```
0
```

Nota:
• Restituisce un intero ≥ 0, usabile in PRINT/LET/IF

BT END

Sintassi:
BT END

Descrizione:
Chiude la UART2 e disattiva la comunicazione BT.

Esempi pratici

```
10 BT END
20 PRINT "CHIUSO"
RUN
```

Output atteso:

```
CHIUSO
```

Nota:
• Usalo prima di cambiare pin/baud con un nuovo BT INIT

BT FLUSH

Sintassi:
BT FLUSH

Descrizione:
Svuota il buffer RX della UART Bluetooth.

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 BT FLUSH
30 BT READ S$
40 PRINT LEN(S$)
RUN
```

Output atteso:

```
0
```

Nota:
• Utile per ripartire “puliti” prima di un nuovo parsing

BT INIT (HC-05/HC-06)

Sintassi:

BT INIT rx tx baud [keyPin]

Descrizione:
Inizializza la UART2 verso il modulo Bluetooth in **data mode** alla velocità baud.
Opzionalmente memorizza keyPin per usare BT ATMODE ON/OFF.

Esempi pratici

Esempio 1 – Init + LED pronto

```
10 PINMODE 32 OUTPUT NOPULL
20 DWRITE 32 0
30 BT INIT 15 12 9600 33
40 PRINT "PRONTO: INVIA ON/OFF"
```

Output atteso:

```
PRONTO: INVIA ON/OFF
```

Esempio 2 – Init senza KEY

```
10 BT INIT 15 12 9600
20 PRINT "BT OK"
```

Output atteso:

```
BT OK
```

Note

- Sintassi: BT INIT rx tx ... → con i tuoi pin è BT INIT 15 12 ...
- Usalo prima di BT SEND/READ/...

BT PRINT

Sintassi:
BT PRINT valore

Descrizione:
Converte valore in testo e lo invia su BT (nessun newline automatico).

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 LET ST=1
30 BT SEND "LED="
40 BT PRINT ST
50 BT SEND "\r\n"
RUN
```

Output atteso:
(sul BT arriva “42”)

Nota:
• Per andare a capo aggiungi BT SEND "\r\n"

BT READ

Sintassi

```
BT READ var$ [n]
```

Descrizione

Legge fino a n byte dal buffer RX.
Se n è omesso, legge tutto ciò che è disponibile.

Esempi

Esempio 1 – Comando singolo per LED

```
10 PINMODE 32 OUTPUT NOPULL

20 DWRITE 32 0

30 BT INIT 15 12 9600 33

40 BT AVAILABLE N

50 IF N=0 THEN GOTO 40

60 BT READ C$ 1

70 IF C$="1" THEN DWRITE 32 1

80 IF C$="0" THEN DWRITE 32 0

90 GOTO 40
```

Esempio 2 – Debug

```
10 BT INIT 15 12 9600 33

20 BT AVAILABLE N

30 IF N=0 THEN GOTO 20

40 BT READ S$

50 PRINT "RX=[";S$;"]"
```

Note

Non attende newline.

BT READLN

Sintassi

BT READLN var$

Descrizione

Legge fino a LF (\n), ignorando \r.
Se \n non è ancora arrivato, restituisce ciò che è presente in quel momento.

Esempio concreto – ON / OFF LED (pattern consigliato)

```
10 PINMODE 32 OUTPUT NOPULL

20 DWRITE 32 0

30 BT INIT 15 12 9600 33

40 PRINT "BT LED READY (ON/OFF)"


50 LET CMD$=""


60 BT AVAILABLE N

70 IF N=0 THEN GOTO 60


80 BT READ T$

90 IF LEN(T$)=0 THEN GOTO 60


100 IF T$=CHR$(13) THEN GOTO 200

110 IF T$=CHR$(10) THEN GOTO 200


120 LET CMD$=CMD$+T$

130 IF LEN(CMD$)>10 THEN LET CMD$=""

140 GOTO 60


200 IF CMD$="ON" THEN GOTO 300
```

```
210 IF CMD$="OFF" THEN GOTO 400
220 BT SEND "CMD ERR\r\n"
230 GOTO 500

300 DWRITE 32 1
310 BT SEND "LED ON\r\n"
320 GOTO 500

400 DWRITE 32 0
410 BT SEND "LED OFF\r\n"
420 GOTO 500

500 LET CMD$=""
510 GOTO 60
```

Note

Nel tuo BASIC è più affidabile accumulare i caratteri e fermarsi su CR/LF.

BT SEND

Sintassi:
BT SEND "testo"
BT SEND var$

Descrizione:
Invia la stringa (letterale o var$) **così com'è** sulla UART Bluetooth.

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 BT SEND "LED ON\r\n"
RUN
```

Output atteso:
(nessun output console; la stringa va su Bluetooth)

Nota:
• Se il peer richiede CRLF, includi \r\n nel testo

BT SETBAUD

Sintassi:
BT SETBAUD rate

Descrizione:
Imposta la **baud dati** del modulo (es. 9600). Tenta la sintassi HC-05 (AT+UART=rate,0,0). Aggiorna la baud usata in modo dati.

Esempi pratici

```
10 BT INIT 15 12 9600 33
20 BT ATMODE ON
30 BT SETBAUD 19200
40 BT ATMODE OFF
50 BT END
60 BT INIT 15 12 19200 33
70 PRINT "BAUD OK"
RUN
```

Output atteso:

```
BAUD OK
```

Nota:
• Su alcuni HC-05 serve AT+RESET per applicare la nuova UART

BT SETNAME

Sintassi:
BT SETNAME "Nome"

Descrizione:
Helper: prova **HC-05** (AT+NAME=Nome) e **HC-06** (AT+NOMENome). Non legge la risposta.

Esempi pratici

```
10 BT SETNAME "BASIC32"
20 BT AT "AT+NAME?" RESP N$ 1000
30 PRINT N$
RUN
```

Output atteso:

```
+NAME:BASIC32
```

Nota:
• Verifica con BT AT … RESP … se vuoi l'eco/OK

BT SETPIN

Sintassi:
BT SETPIN "1234"

Descrizione:
Helper: prova **HC-05** (AT+PSWD=1234) e **HC-06** (AT+PIN1234). Non legge la risposta.

Esempi pratici

```
10 BT SETPIN "1234"
20 BT AT "AT+PSWD?" RESP P$ 1000
30 PRINT P$
RUN
```

Output atteso:

```
+PSWD:1234
```

Nota:
• La sintassi di lettura del PIN può variare con il firmware

BT TIMEOUT

Sintassi:
BT TIMEOUT ms

Descrizione:
Imposta il **timeout di default** (ms) per letture AT (BT AT … RESP …, BT VERSION).

Esempi pratici

```
10 BT TIMEOUT 1500
20 PRINT "TIMEOUT OK"
RUN
```

Output atteso:

```
TIMEOUT OK
```

Nota:
• Influisce solo sulle letture AT con risposta a riga

BT VERSION

Sintassi:
BT VERSION var$

Descrizione:
Tenta AT+VERSION? (HC-05) e, se vuoto, AT+VERSION (HC-06). Scrive la risposta in var$.

Esempi pratici

```
10 BT ATMODE ON
20 BT VERSION V$
21 IF LEN(V$)=0 THEN PRINT "NESSUNA RISPOSTA"
30 PRINT V$
RUN
```

Output atteso:

```
+VERSION:2.0-20100601
```

Nota:
• Richiede AT mode attivo

CALLFUNC

Sintassi:

CALLFUNC <nome>

Descrizione:

Esegue una funzione definita con FUNC, una sola volta.
L'esecuzione è **bloccante**, cioè il programma attende che la funzione finisca prima di proseguire.

Esempi pratici

Esempio 1 – Chiamare una funzione LED

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC LAMP
20 DWRITE 2 1
30 DELAY 500
40 DWRITE 2 0
50 DELAY 500
60 ENDFUNC
70 CALLFUNC LAMP
```

Output atteso:
Il LED si accende e si spegne una volta con 500 ms di attesa.

Esempio 2 – Uso ripetuto della funzione

```
80 CALLFUNC LAMP
90 DELAY 1000
100 GOTO 80
```

Output atteso:
Il LED lampeggia ogni secondo, grazie all'uso ripetuto della funzione.

Note:

La funzione deve essere già definita con FUNC

Il nome deve corrispondere esattamente

Può essere richiamata più volte nel programma

È bloccante: il programma attende che termini

Non funziona con funzioni LOOP (in quel caso usare STARTFUNC)

CARDKB

Funzione numerica — legge un tasto dalla tastiera **CardKB I2C**

Sintassi

CARDKB

Descrizione

CARDKB restituisce il codice ASCII dell'ultimo tasto premuto sulla tastiera **CardKB** collegata via I²C.

Se **nessun tasto** è stato premuto → restituisce **0**.

Se un tasto è disponibile, la funzione restituisce il **codice ASCII** (0–255).

La lettura è **non bloccante** (non ferma il programma).

Questa funzione è pensata per:

controlli di gioco

menu interattivi

input senza usare il comando INPUT

lettura rapida e continua da tastiere compatte I2C

La tastiera deve essere correttamente collegata al bus I2C tramite il comando:

WIRE <sda> <scl>

Valore restituito

Ritorno	Significato
0	Nessun tasto premuto
1–255	Codice ASCII del tasto premuto
27	Tasto ESC (ferma anche il programma se ESC-STOP è attivo)

Esempio semplice

```
10 PRINT "Premi un tasto sulla CardKB"
20 K = CARDKB
30 IF K > 0 THEN PRINT "Codice:", K, "Char:", CHR$(K)
```

```
40 GOTO 20
```

Funzionamento dell’esempio

Il programma stampa un messaggio

Alla riga 20 legge continuamente la tastiera

Quando un tasto viene premuto:

K diventa il codice ASCII

La riga 30 mostra:

il codice numerico

il carattere leggibile (CHR$(K))

Poi torna alla riga 20 e continua il loop

Esempio: uscita con ESC

```
10 PRINT "Premi tasti, ESC per uscire"
20 K = CARDKB
30 IF K = 27 THEN END
40 IF K > 0 THEN PRINT "Hai premuto:", CHR$(K)
50 GOTO 20
```

Note importanti

CARDKB legge **solo un byte** per chiamata. Se tieni premuto un tasto, la tastiera invia ripetizioni (autorepeat).

È compatibile sia con **CardKB**, **PS/2**, sia con tastiere emulate I2C (funziona su tutte perché passa da devPs2HandleChar).

ESC (ASCII 27) attiva anche lo **STOP del programma** se la modalità ESC-STOP è abilitata.

CHAIN

Sintassi:

CHAIN "file"
CHAIN SD "file"
CHAIN SPIFFS "file"
CHAIN <expr$>
CHAIN SD <expr$>
CHAIN SPIFFS <expr$>

(dove file è il nome del listato; se non specifichi estensione viene aggiunto .bas automaticamente)

Descrizione:

Il comando **CHAIN** permette di **caricare e avviare un altro listato BASIC da dentro un listato in esecuzione**, in stile “multiload” (simile al comportamento dei vecchi home computer).

Quando viene eseguito:

- il programma corrente **termina**
- il nuovo file viene caricato **al posto** del programma corrente
- se CHAIN è chiamato **da un listato**, il nuovo programma viene eseguito subito (**RUN automatico**)
- le **variabili NON vengono cancellate** (restano disponibili nel nuovo listato)

Il file può essere caricato sia da **SPIFFS** che da **SD**, e il nome può essere:

- una stringa letterale ("P2")
- una variabile stringa (F$)
- un’espressione stringa ("PART"+X$)

Esempi pratici

Esempio 1 – Passaggio a un altro listato (stesso device di default)

P1.bas

```
10 A=5
20 PRINT "SONO P1 - A=";A
30 CHAIN "P2.bas"
```

P2.bas

```
10 PRINT "SONO P2 - A=";A
20 END
```

Output atteso:

```
SONO P1 - A=5
SONO P2 - A=5
```

Esempio 2 – Caricare da SD

```
10 PRINT "CARICO DA SD..."
20 CHAIN SD "GIOCO2"
```

Output atteso:

- Viene caricato GIOCO2.bas dalla SD e parte subito.

Esempio 3 – Nome file in variabile / espressione

```
10 P$="PARTE"
20 N$="2"
30 P2$=".bas"
40 CHAIN P$+N$+P2$
```

Carica e avvia:

- PARTE2.bas

Note:

- CHAIN **sostituisce** il programma in memoria con quello nuovo (non fa "merge").
- È pensato per dividere programmi lunghi in più file e passare da uno all'altro.
- Se il file non esiste o non è accessibile, viene generato un errore runtime.
- Se non specifichi SD o SPIFFS, viene usato il device di default del sistema.
- Le variabili restano in memoria: utile per passare parametri tra un listato e l'altro (es. LIV=3, SCORE=1200, ecc.).
- Il file caricato dovrebbe contenere righe BASIC numerate (listato standard).

CHR$(x)

Sintassi:

CHR$(codice)

Descrizione:
La funzione CHR$ restituisce il **carattere ASCII** corrispondente al **valore numerico** x (compreso tra 0 e 255).
È spesso usata per costruire stringhe dinamiche o stampare caratteri speciali (es. INVIO, TAB, lettere accentate, ecc.).

Esempi pratici

Esempio 1 – Da numero a carattere

```
10 PRINT CHR$(65)
RUN
```

Output atteso:

```
A
```

Esempio 2 – Stampare lettere dalla A alla Z

```
10 FOR I = 65 TO 90
20 PRINT CHR$(I);
30 NEXT I
RUN
```

Output atteso:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Esempio 3 – Usare CHR$(10) per andare a capo

```
10 PRINT "RIGA1" + CHR$(10) + "RIGA2"
RUN
```

Output atteso:

```
RIGA1
RIGA2
```

(Se il terminale interpreta correttamente il carattere di nuova linea)

Esempio 4 – Inserire un TAB tra due parole

```
10 PRINT "NOME" + CHR$(9) + "VALORE"
RUN
```

Output atteso:

```
NOME    VALORE
```

Esempio 5 – Costruire stringhe da codice

```
10 T$ = CHR$(72) + CHR$(73)
20 PRINT T$
RUN
```

Output atteso:

```
HI
```

Nota:

- CHR$(10) = newline (LF), CHR$(13) = carriage return (CR)
- CHR$(32) = spazio, CHR$(9) = tabulazione
- Funziona bene insieme a ASC, LEFT$, RIGHT$, MID$

CLS

Sintassi:

CLS

Descrizione:
Il comando CLS **pulisce lo schermo** del terminale seriale inviando un certo numero di righe vuote.
È un metodo semplice per “simulare” la pulizia dello schermo, come avveniva nei vecchi ambienti BASIC.

 Non cancella variabili o codice. È solo un comando visivo per l’utente.

Esempi pratici

Esempio 1 – Pulizia dello schermo con messaggio successivo

```
10 CLS
20 PRINT "BENVENUTO NEL SISTEMA"
RUN
```

Output atteso:

(Schermo vuoto)

Esempio 2 – CLS tra due stampe

```
10 PRINT "PRIMA DEL CLS"
20 WAIT 2000
30 CLS
40 PRINT "DOPO IL CLS"
RUN
```

Output atteso:
Visualizza prima un messaggio, poi lo “nasconde” e mostra il secondo.

CLSANSI

Sintassi:

CLSANSI

Descrizione:
Il comando CLSANSI **pulisce lo schermo del terminale** usando il **codice di controllo ANSI ESC[2J**, che è interpretato da terminali moderni compatibili ANSI (come PuTTY, TeraTerm, minicom, ecc.).

È più rapido ed elegante rispetto a CLS, ma funziona solo se il terminale **supporta ANSI escape codes**.

Esempi pratici

Esempio 1 – Pulizia con sequenza ANSI

```
10 CLSANSI
20 PRINT "PRONTO PER L'INPUT"
RUN
```

Output atteso:

(Il terminale viene pulito all’istante)

Esempio 2 – Confronto tra CLS e CLSANSI

```
10 PRINT "USO CLS:"
20 CLS
30 PRINT "FATTO"
40 WAIT 2000
50 PRINT "USO CLSANSI:"
60 CLSANSI
70 PRINT "FINITO"
RUN
```

Output atteso:
Dipende dal terminale; CLSANSI è più "professionale", mentre CLS è più compatibile.

COPY

Sintassi

COPY "nomefile"
COPY variabile$
COPY "*"
COPY SD "nomefile"
COPY SD variabile$
COPY SD "*"
COPY SPIFFS "nomefile"
COPY SPIFFS variabile$
COPY SPIFFS "*"

Descrizione

Il comando **COPY** copia i file tra la **memoria interna SPIFFS** e la **scheda SD**.

- Senza specificare la memoria, copia **dalla SD verso la SPIFFS** (SD → SPIFFS).
- Con SD, copia **dalla SPIFFS verso la SD** (SPIFFS → SD).
- Con SPIFFS, forza la copia **da SD a SPIFFS** (esplicito).
- Accetta:
  - nomi di file tra virgolette ("setup.bas")
  - variabili stringa o espressioni (FILE$, LEFT$(N$,3)+".bas")
  - l'asterisco **"*"** per copiare **tutti i file** dalla sorgente alla destinazione.
- Se il file esiste già nella destinazione, viene **sovrascritto**.

Esempi pratici

Esempio 1 – Copiare da SPIFFS a SD (default SD)

COPY SD "config.bas"

Output atteso:

Copied SPIFFS -> SD: /config.bas

Esempio 2 – Copiare da SD a SPIFFS

COPY "main.bas"

Output atteso:

Copied SD -> SPIFFS: /main.bas

Esempio 3 – Copiare tutti i file

COPY "*"

Output atteso:

Copied SD -> SPIFFS: /file1.bas
Copied SD -> SPIFFS: /file2.bas
...

Esempio 4 – Usare una variabile stringa

10 NAME$ = "init.bas"
20 COPY SD NAME$
30 PRINT "File copiato su SD"

Note

- Se la SD non è montata o il file non è presente, viene mostrato un messaggio di “Skip”.
- L’operazione non è ricorsiva: copia solo i file nella **directory principale**.
- È possibile copiare un singolo file o tutti i file con "*".
- I percorsi vengono normalizzati automaticamente (viene aggiunto / se mancante).

COS(x)

Sintassi:

COS(x)

Descrizione:
La funzione COS(x) restituisce il **coseno dell'angolo x**, dove x è espresso in **gradi** (non in radianti).
Il valore restituito è un numero compreso tra **-1 e 1**, come previsto dalla funzione coseno.

Può essere usata in calcoli matematici, grafici o condizioni.

Se desideri usare radianti, devi convertire manualmente:

COS(x * 180 / PI)

Esempi pratici

Esempio 1 – Coseno di 60 gradi
→ Il coseno di 60° è 0.5:

```
10 PRINT "COS(60) = "; COS(60)
RUN
```

Output atteso:

```
COS(60) = 0.5
```

Esempio 2 – Calcolo del coseno in ciclo
→ Mostra coseno per angoli da 0 a 360° ogni 30°:

```
10 FOR A = 0 TO 360 STEP 30
20 PRINT "COS("; A; ") = "; COS(A)
30 NEXT A
RUN
```

Output atteso:

```
COS(0) = 1
COS(30) = 0.866
...
```

Esempio 3 – Uso in un'espressione con condizione
→ Verifica se il coseno è negativo:

```
10 A = 135
20 IF COS(A) < 0 THEN PRINT "COSENO NEGATIVO"
RUN
```

Output atteso: COSENO NEGATIVO

DATA

Sintassi:

DATA valore1, valore2, valore3, ...

Descrizione:
Il comando DATA serve per **dichiarare una sequenza di valori costanti** (numerici o stringhe) che possono essere letti in seguito con il comando READ.

I DATA non vengono eseguiti direttamente durante il programma, ma fungono da archivio interno. La lettura avviene in ordine sequenziale e può essere **riavviata** con il comando RESTORE.

Esempi pratici

Esempio 1 – Lettura di numeri da DATA
→ Memorizza tre valori e li legge:

```
10 DATA 100, 200, 300
20 READ A, B, C
30 PRINT A, B, C
RUN
```

Output atteso:

```
100 200 300
```

Esempio 2 – Lettura di stringhe da DATA
→ È possibile anche leggere stringhe racchiuse tra virgolette:

```
10 DATA "UNO", "DUE", "TRE"
20 READ A$, B$, C$
30 PRINT A$, B$, C$
RUN
```

Output atteso:

```
UNO DUE TRE
```

Esempio 3 – Lettura progressiva in loop
→ READ può essere usato anche dentro un ciclo:

```
10 DATA 1, 2, 3, 4, 5
20 FOR I = 1 TO 5
30 READ X
40 PRINT "VALORE "; I; ": "; X
50 NEXT I
RUN
```

Output atteso:

VALORE 1: 1
VALORE 2: 2
VALORE 3: 3
VALORE 4: 4
VALORE 5: 5

Esempio 4 – Uso combinato con RESTORE

→ RESTORE permette di riutilizzare i dati da capo:

```
10 DATA 10, 20
20 READ A, B
30 PRINT A, B
40 RESTORE
50 READ C
60 PRINT C
RUN
```

Output atteso:

10 20
10

DATED

Sintassi:
DATED

Descrizione:
Il comando **DATED** restituisce il **giorno** corrente (valore numerico da 1 a 31), secondo l'orologio interno del sistema.
È utile per operazioni condizionali o controlli sulla data.

Esempi pratici

Esempio 1 – Stampare il giorno corrente

```
10 PRINT DATED
RUN
```

Output atteso:

```
15
```

Nota:
• Restituisce un numero intero
• Il valore dipende dalla data impostata o sincronizzata
• Non richiede parametri

DATEM

Sintassi:
DATEM

Descrizione:
Il comando **DATEM** restituisce il **mese** corrente (valore numerico da 1 a 12), secondo l'orologio interno.
Utile per operazioni che dipendono dal periodo dell'anno.

Esempi pratici

Esempio 1 – Stampare il mese corrente

```
10 PRINT DATEM
RUN
```

Output atteso:

```
6
```

Nota:
• Restituisce un numero intero
• Il valore dipende dalla data impostata o sincronizzata
• Non richiede parametri

DATEY

Sintassi:
DATEY

Descrizione:
Il comando **DATEY** restituisce l'**anno** corrente (es. 2025), secondo l'orologio interno. Consente di ottenere l'anno per controlli, logiche temporali o etichette automatiche.

Esempi pratici

Esempio 1 – Stampare l'anno corrente

```
10 PRINT DATEY
RUN
```

Output atteso:

```
2025
```

Nota:
• Restituisce un numero intero a quattro cifre
• Il valore dipende dalla data impostata o sincronizzata
• Non richiede parametri

DEBUGMEM

Sintassi:

DEBUGMEM

Descrizione:

Il comando DEBUGMEM stampa sul monitor seriale **informazioni dettagliate sullo stato della memoria e delle risorse attive** nel sistema.

È utile per fare debug, analizzare consumi di memoria, verificare perdite, stack residuo, variabili attive e sprite ancora in uso.

Dati visualizzati:

- Quantità di **heap disponibile** (RAM libera)
- **Stack residuo** del task corrente (se su ESP32)
- Numero di variabili **numeriche** e loro valore
- Numero di variabili **stringa** e loro contenuto
- Numero di **array** definiti
- **Sprite attivi** con:
  - ID
  - posizione (X, Y)
  - nome dati associati (es. HEART)

Esempi pratici

Esempio 1 – Debug all'avvio

```
10 DEBUGMEM
RUN
```

Output atteso nel monitor seriale:

```
==== DEBUG MEMORIA ====
Heap disponibile: 128400 byte
Stack residuo task attuale: 3792 byte
Variabili numeriche: 0
Variabili stringa: 0
Array definiti: 0
Sprite attivi: 0
=======================
```

Esempio 2 – Debug dopo esecuzione animazioni

```
10 GOSUB 1000
20 DEBUGMEM
RUN
```

Output:
Mostra lo stato di sprite, variabili e RAM **dopo la funzione grafica**, utile per capire cosa resta in memoria.

Nota:

- DEBUGMEM **non modifica nulla**, è solo diagnostico.
- Se usato in un ciclo, può aiutare a individuare perdite di memoria nel tempo.
- Utile durante lo sviluppo per evitare **crash da overflow o heap pieno**.

DEF FN

Sintassi:

DEF FNnome(arg1, arg2, ...) = espressione

Descrizione:
DEF FN consente di **definire una funzione personalizzata** all'interno del programma.
La funzione prende uno o più **argomenti** e restituisce il valore di una **espressione**.

È utile per **riutilizzare operazioni matematiche** o formule complesse senza doverle scrivere più volte.
Le funzioni devono avere un nome che inizia con FN (es: FNADD, FNSQUARE).

 Non possono contenere comandi interattivi o comandi di controllo (es. PRINT, GOTO, IF, ecc.): solo espressioni.

Esempi pratici

Esempio 1 – Funzione somma a due argomenti
→ Definisce una funzione che restituisce la somma di due numeri:

```
10 DEF FNADD(X, Y) = X + Y
20 PRINT "5 + 3 = "; FNADD(5, 3)
RUN
```

Output atteso:

```
5 + 3 = 8
```

Esempio 2 – Calcolo del quadrato
→ Funzione per elevare un numero al quadrato:

```
10 DEF FNSQ(X) = X * X
20 PRINT "7^2 = "; FNSQ(7)
RUN
```

Output atteso:

```
7^2 = 49
```

Esempio 3 – Uso con variabili e formule
→ Una funzione per la formula dell'area del cerchio:

```
10 DEF FNAREA(R) = 3.14 * R * R
20 INPUT R
30 PRINT "AREA = "; FNAREA(R)
RUN
```

Output atteso (se inserisci 2):

AREA = 12.56

Esempio 4 – Richiamo multiplo

→ Una funzione può essere richiamata più volte:

```
10 DEF FNTRIPLA(X) = X * 3
20 FOR I = 1 TO 5
30 PRINT "TRIPLO DI "; I; " = "; FNTRIPLA(I)
40 NEXT I
RUN
```

Output atteso:

```
TRIPLO DI 1 = 3
TRIPLO DI 2 = 6
TRIPLO DI 3 = 9
TRIPLO DI 4 = 12
TRIPLO DI 5 = 15
```

DEL

Sintassi:

DEL "<nomefile>"
DEL variabile$
DEL SD "<nomefile>"
DEL SD variabile$
DEL SPIFFS "<nomefile>"
DEL SPIFFS variabile$

Descrizione:

Il comando **DEL** elimina un file dalla memoria del sistema.

- Se eseguito **senza specificare la memoria**, il file viene cancellato dalla **memoria interna SPIFFS**.
- Specificando **SD**, il file viene eliminato dalla **scheda SD**.
- È possibile indicare esplicitamente **SPIFFS** per maggiore chiarezza, ma non è obbligatorio.
- Il nome del file può essere indicato **direttamente tra virgolette** oppure **tramite una variabile stringa** (es. FN$).

Questo comando è utile per gestire i file presenti nelle diverse memorie dell'ESP32, senza ambiguità.

Esempi pratici

Esempio 1 – Eliminare un file da SPIFFS

```
DEL "main.bas"
```

Esempio 2 – Eliminare un file da SD

```
DEL SD "demo.bas"
```

Esempio 3 – Usare una variabile per specificare il file da cancellare

```
10 LET FN$ = "prog1.bas"
20 DEL FN$
RUN
```

Esempio 4 – Eliminare tramite variabile da SD

```
10 LET FN$ = "test.bas"
20 DEL SD FN$
RUN
```

Output atteso:

File deleted from SPIFFS.

oppure

File deleted from SD.

Nota:

- DEL accetta un **solo argomento opzionale iniziale**: SD o SPIFFS.
- È possibile usare **variabili stringa** per specificare il nome del file.
- Se la scheda SD non è collegata o montata, DEL SD restituirà un errore di eliminazione fallita.
- Se il file non esiste, viene segnalato un errore.

DELN

Sintassi

```
DELN n
DELN n1,n2,n3
DELN a-b
```

Descrizione

Il comando **DELN** elimina una o più righe del programma in memoria.

- Può rimuovere una singola riga, più righe separate da virgole, o un intervallo continuo.
- Se viene digitato **solo il numero di riga** (senza la parola chiave DELN), la riga viene eliminata comunque.
- Se una riga specificata non esiste, viene ignorata senza generare errori.

Questa funzione è utile per modificare rapidamente il listato senza doverlo riscrivere da capo.

Esempi pratici

Esempio 1 – Rimuovere una riga specifica

```
10 PRINT "Ciao"
20 PRINT "Riga da cancellare"
30 PRINT "Fine"
RUN
```

Poi:

```
DELN 20
LIST
```

Output atteso:

```
10 PRINT "Ciao"
30 PRINT "Fine"
```

Esempio 2 – Eliminare più righe in una volta

```
10 PRINT "Riga1"
20 PRINT "Riga2"
30 PRINT "Riga3"
```

```
40 PRINT "Riga4"
```

Comando:

```
DELN 10,30
LIST
```

Output atteso:

```
20 PRINT "Riga2"
40 PRINT "Riga4"
```

Esempio 3 – Eliminare un intervallo di righe

```
100 PRINT "A"
110 PRINT "B"
120 PRINT "C"
130 PRINT "D"
```

Comando:

```
DELN 100-120
LIST
```

Output atteso:

```
130 PRINT "D"
```

Esempio 4 – DELN su riga inesistente

```
DELN 500
```

Se la riga 500 non è presente, il programma resta invariato e non si genera nessun errore.

Note

- DELN agisce **solo sul listato in memoria**.
- Per rendere permanenti le modifiche, usa SAVE "nomefile.bas".
- Funziona sia con singole righe, che con **liste di righe** (10,20,30) e **intervalli** (100-200).
- Usalo insieme a EDIT, LIST, NEW, SAVE, LOAD per la gestione completa dei programmi BASIC.

DELVAR

Nome

DELVAR

Sintassi

DELVAR "file"
DELVAR "file" "chiave"

DELVAR SPIFFS "file"
DELVAR SPIFFS "file" "chiave"

DELVAR SD "file"
DELVAR SD "file" "chiave"

Descrizione

DELVAR elimina dati da un file JSON.

- Se specifichi solo "file" → elimina **tutto il file**
- Se specifichi "file" "chiave" → elimina solo la **chiave** indicata, lasciando intatte le altre

Esempi

Esempio 1 – Eliminare una chiave da SPIFFS

10 DELVAR SPIFFS "prefs.json" "USER"
20 LISTVARS SPIFFS "prefs.json"

Esempio 2 – Cancellare completamente un file su SD

10 DELVAR SD "dati.json"

Note

- Se il file non esiste, la cancellazione del file non produce errore.
- Se la chiave non esiste, il file resta invariato.
- La cancellazione del file è irreversibile.

DELAY n

Sintassi:

DELAY n

Descrizione:
Il comando DELAY sospende l'esecuzione del programma per n **millisecondi**.
Durante il ritardo, il microcontrollore **non esegue altro codice**: è una pausa **bloccante**.

È utile per:

- Attendere tra due operazioni
- Creare animazioni o lampeggi
- Simulare tempi di caricamento

Esempi pratici

Esempio 1 – Pausa tra due messaggi

```
10 PRINT "CIAO"
20 DELAY 1000
30 PRINT "MONDO"
RUN
```

Output atteso (con 1 secondo di pausa tra le due righe):

```
CIAO
(monitora pausa)
MONDO
```

Esempio 2 – Lampeggio simulato

```
10 CLS
20 PRINT "☼"
30 DELAY 500
40 CLS
50 DELAY 500
60 GOTO 10
RUN
```

Output atteso:
Un lampeggio infinito del simbolo "☼" ogni mezzo secondo.

Esempio 3 – Ritardo dopo lettura

```
10 INPUT "INSERISCI IL TUO NOME: "; N$
20 PRINT "ATTENDI..."
30 DELAY 2000
40 PRINT "BENVENUTO "; N$
RUN
```

Output atteso:
Dopo l’input, una pausa di 2 secondi prima del messaggio di benvenuto.

Esempio 4 – Conto alla rovescia

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 DELAY 1000
40 NEXT I
50 PRINT "VIA!"
RUN
```

Output atteso:
Un countdown da 5 a 1 con 1 secondo tra ciascun numero.

Nota:

- DELAY blocca **totalmente** l’esecuzione (incluso INPUT, GET, ecc.)
- L'unità di misura è il **millisecondo** (1000 = 1 secondo)

DEV AUTO ON/OFF

Sintassi:

DEV AUTO ON
DEV AUTO OFF

Descrizione:
Imposta se la modalità developer deve avviarsi automaticamente al boot dell'ESP32. Quando è attiva, al riavvio il display configurato e l'eventuale tastiera PS/2 vengono inizializzati senza dover reinserire i comandi manualmente.

Esempi pratici:

```
10 DEV AUTO ON
20 DEV ILI SET 17 16 5 32 18 23 19 3
30 DEV DISPLAY ILI
40 DEV ON
RUN
```

→ Alla successiva accensione l'ESP32 partirà direttamente in modalità developer con display pronto.

DEV CLEAR

Sintassi:

DEV CLEAR

Descrizione:
Cancella lo schermo del display attualmente selezionato in modalità developer.
Non disattiva il dev mode, ma pulisce solo il contenuto.

Esempio pratico:

DEV CLEAR

→ Cancella la console sul display (utile per avere una schermata pulita).

DEV DISPLAY

Sintassi

DEV DISPLAY ILI

Descrizione

Seleziona quale **display** usare come uscita della console DEV (per tutti i comandi di stampa: PRINT, LIST, EDIR, messaggi, ecc.).
La selezione viene **salvata** in configurazione e rimane attiva ai successivi avvii.

Nota: il display scelto deve essere **già configurato** (pin/indirizzi, modalità, ecc.). In caso contrario viene segnalato **errore** e la selezione non cambia.

Esempio – Seleziona ILI9341

```
10 DEV DISPLAY ILI
20 DEV CLEAR
30 PRINT "Schermo ILI attivo"
RUN
```

Note

- Puoi passare da un display all’altro con lo stesso comando: DEV DISPLAY ILI.
- Se il dispositivo non è configurato o non inizializza, il comando segnala errore e resta il display precedente.
- **Prestazioni:** per stampe molto lunghe attiva DEV SERIAL OFF per massima reattività.
- **Colonne/Righe:** possono variare a seconda del display e della sua risoluzione/impostazioni; usa i tuoi comandi dedicati (COLONNE/RIGHE, MAPCOLS/MAPROWS) se presenti per adattare il layout.

DEV FONT

Sintassi:
DEV FONT <size>
Descrizione:
Imposta il fattore di scala del font nella console DEV.
Esempi pratici

DEV FONT 1
DEV FONT 2

Nota:
• Valore minimo 1.
• Il cambio di font effettua un clear del display.

DEV ILI SET

Sintassi:

DEV ILI SET <CS> <DC> <RST> <LED> [SCK MOSI MISO] [ROT]

Descrizione:
Configura un display ILI9341 specificando i pin di collegamento.
L'argomento LED indica il pin di controllo retroilluminazione.
Parametri opzionali: SCK MOSI MISO per bus SPI custom, ROT per rotazione.

Esempio pratico:

DEV ILI SET 17 16 5 32 18 23 19 3
DEV DISPLAY ILI
DEV ON

→ Inizializza un ILI9341 con CS=17, DC=16, RST=5, retroilluminazione sul GPIO32.

DEV MAPCOLS

Sintassi:

DEV MAPCOLS <n>

Descrizione:
Imposta il numero di colonne logiche della console developer.
Serve per mappare correttamente la stampa del testo su display grafici.

Esempio pratico:

DEV MAPCOLS 40

→ La console developer userà 40 colonne virtuali.

DEV MAPROWS

Sintassi:

DEV MAPROWS <n>

Descrizione:
Imposta il numero di righe logiche della console developer.

Esempio pratico:

DEV MAPROWS 20

→ La console avrà 20 righe virtuali.

DEV ON / OFF

Sintassi:

```
DEV ON
DEV OFF
```

Descrizione:
Gestisce lo stato della modalità developer.

- DEV ON → attiva la console developer e duplica tutti i Serial.print anche sul display selezionato.
- DEV OFF → disattiva la console developer, lasciando attiva solo la seriale.

Esempi pratici:

```
DEV DISPLAY ILI
DEV ON
```

→ Attiva la console su ILI e tutti i messaggi saranno visibili anche lì.

→ Riavvia automaticamente ESP per inizializzare la configurazione.

```
DEV OFF
```

→ Disattiva la console su display, continuando a lavorare solo da seriale.

DEV PS2 LAYOUT

Sintassi

DEV PS2 LAYOUT <US | IT | DE | FR | ES>

Descrizione

Imposta il layout della tastiera PS/2, permettendo la corretta mappatura dei tasti in base al paese.

Layout supportati:

- **US** → Americano (default)
- **IT** → Italiano
- **DE** → Tedesco
- **FR** → Francese
- **ES** → Spagnolo

Esempi

DEV PS2 LAYOUT IT

Oppure:

DEV PS2 LAYOUT US

Note

- Il layout viene salvato in prefs.cfg e ripristinato automaticamente al riavvio.
- Può essere modificato anche con la tastiera PS/2 già in funzione.
- È necessario aver configurato i pin (con DEV PS2 SET) per usare la tastiera.

DEV PS2 OFF

Sintassi

DEV PS2 OFF

Descrizione

Disattiva la tastiera PS/2 in modalità developer.
Quando è OFF, i tasti premuti non vengono letti né inviati alla console DEV.

Esempio

DEV PS2 OFF

Note

- Interrompe immediatamente la lettura della tastiera PS/2.
- La configurazione dei pin rimane memorizzata, ma la tastiera resta inattiva finché non viene riabilitata con DEV PS2 ON.

DEV PS2 ON

Sintassi

DEV PS2 ON

Descrizione

Attiva la tastiera PS/2 per l’uso nella console DEV.
Permette di digitare comandi BASIC direttamente dal display ILI in tempo reale.

Esempio

DEV PS2 ON

Note

- **Richiede una precedente configurazione dei pin** tramite DEV PS2 SET.
- Se i pin non sono stati configurati, il comando genera errore.

DEV PS2 SET

Sintassi

DEV PS2 SET <dataPin> <clkPin>

Descrizione

Configura i pin utilizzati dalla tastiera PS/2.
Dopo aver impostato i pin, la tastiera può essere usata per digitare direttamente comandi BASIC sulla console DEV (display ILI).

Attenzione:
La sintassi reale del firmware è DATA → CLK (in questo ordine).

Esempio

DEV PS2 SET 26 27

Effetto

DATA = 26
CLK = 27

La tastiera PS/2 viene collegata su DATA=26 e CLK=27 ed è pronta per essere abilitata con DEV PS2 ON.

Note

- La configurazione dei pin viene salvata in prefs.cfg e ricaricata automaticamente all'avvio.
- Permette alla tastiera di funzionare in tempo reale sulla console DEV.

DEV SERIAL ON / OFF

Sintassi

DEV SERIAL ON
DEV SERIAL OFF

Descrizione

Abilita o disabilita il **mirroring** dell'output della console DEV sulla porta **Serial** (USB).

- Con DEV SERIAL ON, tutto ciò che normalmente stampa sul display (es. PRINT, LIST, messaggi di stato) viene **replicato anche sul Serial Monitor**.
- Con DEV SERIAL OFF, l'output **non** viene replicato su Serial: utile per **massima velocità** su ILI/OLED durante stampe lunghe (es. LIST, directory).

Non modifica il **baud rate** né le impostazioni della seriale; controlla solo se inviare o meno le stampe su Serial.
L'impostazione **non è persistente**: dopo il riavvio torna al valore predefinito (di solito **OFF**). Se vuoi mantenerla, metti il comando nel tuo script di avvio.

Esempi pratici

Esempio 1 – Attiva log USB

```
10 DEV SERIAL ON
20 PRINT "Ciao sia su schermo che su USB!"
RUN
```

Mostra il testo sul display selezionato (ILI) **e** sul Serial Monitor.

Esempio 2 – LIST veloce (senza eco su USB)

```
10 DEV SERIAL OFF
20 LIST
RUN
```

LIST risulta molto più reattivo su ILI perché evita il costo delle stampe su USB.

Esempio 3 – Log USB solo durante una sezione

```
10 DEV SERIAL OFF
20 PRINT "Stampa veloce sul display"
30 DEV SERIAL ON
```

```
40 PRINT "Questa riga va anche su USB"
50 DEV SERIAL OFF
60 PRINT "Di nuovo solo display"
RUN
```

Esempio 4 – Debug condizionale

```
10 D=1
20 IF D THEN DEV SERIAL ON
30 PRINT "Messaggi di debug"
40 IF D THEN DEV SERIAL OFF
50 PRINT "Uscita silenziosa su USB"
RUN
```

Note

- **Performance:** DEV SERIAL OFF è consigliato durante stampe lunghe (es. LIST, elenco file) per evitare rallentamenti dovuti al buffer USB/Serial.
- **Ambito:** vale per **tutti** i display supportati e non influisce sull'input da tastiera/PS2 o su altre periferiche.
- **Persistenza:** l'impostazione non viene salvata in config.
- **Compatibilità:** non cambia Serial.begin(...); usa il baud già impostato nel firmware. Se il Serial Monitor non è aperto, DEV SERIAL ON non causa errori, ma l'output può comunque accumularsi nel buffer USB.

DEV STATUS

Sintassi:
DEV STATUS
Descrizione:
Mostra lo stato corrente della modalità developer (attivo/auto, display selezionato, font, mappa, cursore, PS/2).
Esempi pratici

DEV STATUS

Nota:
• Mostra anche le configurazioni salvate per ILI.

DHTCALIBRESET

Sintassi:

- DHTCALIBRESET pin → resetta tutte le calibrazioni associate a quel pin (DHT, DHTT, DHTH).
- DHTCALIBRESET ALL → resetta tutte le calibrazioni di tutti i pin.

Descrizione:
Cancella le impostazioni di calibrazione e ripristina i valori di default:

- offset = 0
- slope = 1

Può essere usato quando vuoi azzerare correzioni precedentemente impostate con DHTCALIBSET.

Esempi pratici

Esempio 1 – Reset calibrazione su pin 2

```
10 DHTCALIBSET DHT 2 0.5 -2
20 DHTCALIBRESET 2
30 SDHCALIBSHOW
RUN
```

Output atteso:

```
(nessuna calibrazione attiva per pin 2)
```

Esempio 2 – Reset totale di tutte le calibrazioni

```
10 DHTCALIBSET DHT 2 1.0
20 DHTCALIBSET DHT 4 -0.5
30 DHTCALIBRESET ALL
40 SDHCALIBSHOW
RUN
```

Output atteso:

```
(nessuna calibrazione attiva)
```

Esempio 3 – Dopo reset lettura senza correzione

```
10 DHTCALIBSET DHTT 2 1.0
20 DHTREAD 2 T H
30 PRINT T
40 DHTCALIBRESET 2
50 DHTREAD 2 T H
```

```
60 PRINT T
RUN
```

Output atteso (esempio):

```
T=(temperatura+1.0)    // prima della reset
T=(temperatura_grezza) // dopo reset
```

Nota:

- pin = numero del pin usato nel DHTINIT.
- Con ALL elimina tutte le calibrazioni registrate.
- Dopo reset, le letture ritornano ai valori “grezzi” del sensore.

DHTCALIBSET

Sintassi:

Forma semplice

- DHTCALIBSET DHT pin offset [slope] → applica **a Temperatura e Umidità**
- DHTCALIBSET DHTT pin offset [slope] → applica **solo alla Temperatura**
- DHTCALIBSET DHTH pin offset [slope] → applica **solo all'Umidità**

Forma estesa

- DHTCALIBSET DHT pin t_offset h_offset [t_slope] [h_slope]

Descrizione:
Imposta la **calibrazione** delle letture DHT sul pin.
Le correzioni seguono la formula:
valore_calibrato = valore_grezzo * slope + offset

- offset: correzione additiva (°C per T, %RH per H)
- slope: fattore moltiplicativo (predefinito 1.0, deve essere **> 0** e ragionevole)
- Priorità in lettura (DHTREAD): **DHTT/DHTH** (specifiche) sovrascrivono **DHT** (generale).

Esempi pratici

Esempio 1 – Offset distinti per T e H (forma estesa, 1 riga)

```
10 DHTCALIBSET DHT 17 0.5 -2
20 DHTINIT 17 11
30 DHTREAD 17 T H
40 PRINT T; "C  "; H; "%"
RUN
```

Output atteso (esempio):

(Temperatura_grezza + 0.5) (Umidità_grezza - 2)

Esempio 2 – Stessa calibrazione su T e H (forma semplice)

```
10 DHTCALIBSET DHT 17 0.3
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT "T=";T;"  H=";H
RUN
```

Output atteso:

T=(T_grezza + 0.3) H=(H_grezza + 0.3)

Esempio 3 – Solo Temperatura con slope

```
10 DHTCALIBSET DHTT 17 0.0 1.02
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT "T=";T
RUN
```

Output atteso:

```
T=(T_grezza * 1.02)
```

Esempio 4 – Solo Umidità, offset -1.5 (slope implicito = 1)

```
10 DHTCALIBSET DHTH 17 -1.5
20 DHTINIT 17 22
30 DHTREAD 17 T H
40 PRINT H
RUN
```

Output atteso:

```
H=(H_grezza - 1.5)
```

Esempio 5 – Validazione slope (errore se negativo o non ragionevole)

```
10 DHTCALIBSET DHT 17 0.5 -2
RUN
```

Output atteso:

```
DHTCALIBSET: Invalid slope (must be > 0 and reasonable)
```

Nota:
• **Slope** è opzionale; default 1.0. Deve essere **> 0** (es. valori tipici 0.9…1.1).
• La **forma estesa** (DHT pin t_off h_off [t_slope] [h_slope]) è la più chiara quando vuoi correzioni diverse su T e H in **un'unica riga**.
• Le calibrazioni restano attive finché non le cambi o esegui un reset (es. DHTCALIBRESET pin o DHTCALIBRESET ALL).
• Compatibile con PRINT, LET, IF perché agisce "a monte" delle variabili lette con DHTREAD.
• In DHTREAD le calibrazioni **specifiche** (DHTT:pin, DHTH:pin) hanno priorità sulla calibrazione **generale** (DHT:pin).

DHTCALIBSHOW

Sintassi:
DHTCALIBSHOW

Descrizione:
Mostra a schermo le calibrazioni DHT correnti (chiavi e parametri offset/slope) per tutti i pin configurati.

Esempi pratici

Esempio 1 – Elenco calibrazioni attive

```
10 DHTCALIBSHOW
RUN
```

Output atteso:

```
DHT:2  => offset=0.500 slope=1.000
DHTT:2 => offset=0.000 slope=1.020
DHTH:2 => offset=-1.500 slope=1.000
```

Esempio 2 – Dopo reset calibrazione

```
10 DHTCALIBSET DHT 2 0 1
20 DHTCALIBSHOW
RUN
```

Output atteso:

```
DHT:2 => offset=0.000 slope=1.000
```

Esempio 3 – Stampa condizionale (se serve)

```
10 DHTCALIBSHOW
20 PRINT "FINE"
RUN
```

Output atteso:

```
DHT:... => offset=... slope=...
FINE
```

Nota:
• È di sola lettura: **non modifica** le impostazioni.
• Utile per debug prima di acquisire misure.

DHTINIT (DHT11-DHT22)

Sintassi:
DHTINIT pin tipo

Descrizione:
Inizializza un sensore **DHT** collegato al **pin** indicato.
tipo = 11 (DHT11) oppure 22 (DHT22/AM2302).
Deve essere eseguito **prima** di DHTREAD.

Esempi pratici

Esempio 1 – Inizializzare un DHT22 su pin 2

```
10 DHTINIT 2 22
20 PRINT "OK"
RUN
```

Output atteso:

```
OK
```

Esempio 2 – Variabili per pin e tipo

```
10 LET P = 7
20 LET T = 11
30 DHTINIT P T
40 PRINT "DHT PRONTO"
RUN
```

Output atteso:

```
DHT PRONTO
```

Esempio 3 – Re-inizializzazione su altro pin

```
10 DHTINIT 2 22
20 DHTINIT 4 22
30 PRINT "SPOSTATO SU PIN 4"
RUN
```

Output atteso:

```
SPOSTATO SU PIN 4
```

Nota:
• Va chiamato almeno una volta prima di leggere.
• Se cambi pin o modello, ripeti DHTINIT.
• Restituisce 1/OK (se la tua piattaforma lo prevede) oppure nessun valore.

DHTREAD

Sintassi:
DHTREAD pin varTemp varHum

Descrizione:
Legge **temperatura (°C)** e **umidità (%)** dal DHT inizializzato su pin e salva i valori in varTemp e varHum.
Se una variabile termina con $, il valore viene salvato come **stringa**; altrimenti come **numero**.
Applica automaticamente eventuali **calibrazioni** impostate con DHTCALIBSET.

Esempi pratici

Esempio 1 – Lettura base

```
10 DHTINIT 2 22
20 DHTREAD 2 T H
30 PRINT "T=";T;"C  H=";H;"%"
RUN
```

Output atteso (esempio):

```
T=24.0C  H=48.0%
```

Esempio 2 – Uso con stringhe

```
10 DHTINIT 2 22
20 DHTREAD 2 T$ H$
30 PRINT "T=";T$;"  H=";H$;"%"
RUN
```

Output atteso (esempio):

```
T=24.0  H=48.0%
```

Esempio 3 – Condizione su umidità

```
10 DHTINIT 2 22
20 DHTREAD 2 T H
30 IF H > 70 THEN PRINT "UMIDITA' ALTA"
RUN
```

Output atteso:

```
UMIDITA' ALTA
```

Nota:
• Richiede DHTINIT eseguito per quel pin.
• In caso di errore lettura alcune implementazioni producono NaN/valore sentinella: gestiscilo con IF.

DIM

Sintassi (classica)

DIM nome_array(n)
DIM nome_array$(n)

Descrizione

DIM dichiara un **array monodimensionale** (vettore), indicizzato da **0 a n (incluso)**.

- **Array numerici**: senza $
  Esempio: DIM A(5) → crea A(0)…A(5) (6 elementi), inizializzati a 0.
- **Array di testo**: con $
  Esempio: DIM NOME$(5) → crea NOME$(0)…NOME$(5) (6 elementi), inizializzati a "".

L'assegnazione dei valori deve essere effettuata con **LET**:

LET A(0) = 10
LET NOME$(1) = "LUCA"

Regole generali

- Indici validi: **0..n** (incluso).
- Gli indici possono essere costanti o variabili numeriche.
- Accesso fuori limite → errore: **INDEX OUT OF RANGE**.
- Non puoi cambiare la dimensione di un array già dichiarato: usa prima **CLEAR ARRAY**.
- Gli array numerici e di testo sono indipendenti: puoi avere A() e A$() contemporaneamente.

Esempi pratici (classici)

Esempio 1 – Array numerico

```
10 DIM A(3)
20 LET A(0) = 5
30 LET A(1) = 10
40 LET A(2) = 15
50 FOR I = 0 TO 2
60 PRINT "A("; I; ") = "; A(I)
70 NEXT I
```

Output atteso

A(0) = 5
A(1) = 10

A(2) = 15

Esempio 2 – Array di testo

```
10 DIM NOME$(2)
20 LET NOME$(0) = "LUCA"
30 LET NOME$(1) = "ANNA"
40 FOR I = 0 TO 1
50 PRINT "NOME("; I; ") = "; NOME$(I)
60 NEXT I
```

Output atteso

```
NOME(0) = LUCA
NOME(1) = ANNA
```

Esempio 3 – Calcolo in ciclo

```
10 DIM N(5)
20 FOR I = 0 TO 5
30   LET N(I) = I * I
40 NEXT I
50 PRINT "N(3) = "; N(3)
```

Output atteso

```
N(3) = 9
```

Esempio 4 – Concatenazione stringhe

```
10 DIM MSG$(2)
20 LET MSG$(0) = "CIAO"
30 LET MSG$(1) = MSG$(0) + " MONDO"
40 PRINT MSG$(1)
```

Output atteso

```
CIAO MONDO
```

Estensioni di DIM — Elenco file

Sintassi (senza virgole)

```
DIM FILE   buf$ pos() len() size() contatore
DIM FILESD buf$ pos() len() size() contatore
```

Descrizione

- **FILE** → carica l’elenco dei file dalla memoria **SPIFFS**.

- **FILESD** → carica l'elenco dei file dalla **scheda SD** (**errore se SD non inizializzata**).
- buf$: variabile stringa che conterrà **tutti i nomi concatenati** (senza separatori).
- pos() : array numerico con la **posizione 1-based** di inizio di ciascun nome dentro buf$.
- len() : array numerico con la **lunghezza** di ciascun nome.
- size() : array numerico con la **dimensione in byte** di ciascun file.
- contatore : **nome variabile numerica a tua scelta** (es. NF, NFILES, TOT) in cui viene scritto **il numero di file caricati**.

Importante: pos(), len(), size() devono essere **già dichiarati** con DIM prima di usare DIM FILE….
Se le loro capienze differiscono, viene usata la **capacità minima** tra i tre.
Gli elementi riempiti sono agli indici **1..contatore** (1-based per comodità d'uso), mentre i tuoi array restano comunque allocati 0..n.

Errori possibili

- **UNDEFINED ARRAY** → pos(), len() o size() non sono stati dichiarati.
- **SD NOT INITIALIZED** → DIM FILESD … usato ma la SD non è montata/pronta.
- **INDEX OUT OF RANGE** → regola generale sugli accessi agli array.

Esempi pratici (file)

Esempio A – Elenco completo da SPIFFS

```
10 DIM POS(512)
20 DIM LENN(512)
30 DIM SIZE(512)
40 BUF$=""
50 DIM FILE BUF$ POS() LENN() SIZE() NF
60 IF NF=0 THEN PRINT "Nessun file in SPIFFS"
70 FOR I=1 TO NF
80   N$ = MID$(BUF$, POS(I), LENN(I))
90   PRINT N$; "  "; SIZE(I); " B"
100 NEXT I
```

Esempio B – SD (già montata) e stampa di un solo file

```
10 DIM POS(512)
20 DIM LEN(512)
30 DIM SIZE(512)
40 BUF$=""
50 DIM FILESD BUF$ POS() LEN() SIZE() NFILES
60 IF NFILES=0 THEN PRINT "Nessun file su SD"
70 K = 2
80 N$ = MID$(BUF$, POS(K), LEN(K))
```

```
90 PRINT "File #"; K; ": "; N$; "  "; SIZE(K); " B"
```

Esempio C – Ricerca di un nome file

```
10 DIM POS(512): DIM LEN(512): DIM SIZE(512)
20 BUF$=""
30 DIM FILE BUF$ POS() LEN() SIZE() NF
40 IF NF=0 THEN PRINT "SPIFFS vuoto"
50 INPUT "Nome da cercare"; T$
60 FOUND = 0
70 FOR I=1 TO NF
80   N$ = MID$(BUF$, POS(I), LEN(I))
90   IF N$ = T$ THEN FOUND = I : GOTO 120
100 NEXT I
110 PRINT "File non trovato": END
120 PRINT "Trovato: "; N$; "  "; SIZE(FOUND); " B"
```

Note tecniche

- Il **contatore** (ultimo parametro in DIM FILE…) è una **variabile numerica arbitraria**: la decidi tu (es. NF, NFILES, TOT). Non è riservata.
- Il DIM **classico** non crea/aggiorna contatori: se vuoi usarne uno, lo gestisci tu (LET K=…) oppure usi quello passato a DIM FILE….
- Le posizioni in pos() sono **1-based** per semplificare l'uso con MID$.
- I nomi file in buf$ sono **concatenati senza separatori**: per leggerli usa sempre MID$(buf$, pos(i), len(i)).

DLEVEL(p)

Sintassi:

DLEVEL(pin)

Descrizione:
Restituisce **sempre il livello digitale "raw"** del pin, **ignorando** qualsiasi debounce software.
Usala quando vuoi conoscere lo **stato istantaneo** del GPIO (utile per rilevare "tenuto premuto", durata della pressione, ecc.).

Restituisce:

- 1 se il pin è **ALTO** (HIGH, ~3.3V)
- 0 se il pin è **BASSO** (LOW, ~0V)

Configura prima il pin con PINMODE (INPUT + eventuale PULLUP/PULLDOWN/NOPULL).

Esempi pratici

Esempio 1 – Lettura diretta

```
10 PINMODE 12 INPUT NOPULL
20 V = DLEVEL(12)
30 PRINT "PIN 12 = "; V
```

Output atteso: 0 oppure 1 in base al segnale.

Esempio 2 – Pulsante con PULLUP (stato tenuto)

```
10 PINMODE 12 INPUT PULLUP
20 IF DLEVEL(12) = 0 THEN PRINT "PREMUTO" ELSE PRINT "RILASCIATO"
30 DELAY 500
40 GOTO 20
```

Comportamento: stampa continuamente lo stato reale (0=premuto, 1=rilasciato).

Esempio 3 – Con DEBOUNCE attivo (dimostrazione che lo ignora)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "HOLD" ELSE PRINT "UP"
30 DELAY 500
40 GOTO 20
```

Comportamento: DLEVEL mostra lo stato in tempo reale; il debounce non ha effetto su DLEVEL.

Note

- Con **PULLUP**: **premuto = 0**, **rilasciato = 1**. Con **PULLDOWN** è l'inverso.
- **Differenza da DREAD**:
  - DREAD(pin) può diventare **one-shot** se abiliti DEBOUNCE in PINMODE.
  - DLEVEL(pin) è **sempre raw**, indipendente da DEBOUNCE.
- Per contare click singoli usa DREAD con DEBOUNCE; per rilevare "hold" o tempi di pressione usa DLEVEL.

DO

Sintassi:
DO <numero_di_linea>

Descrizione:
Il comando DO permette di eseguire ripetutamente una singola riga di programma ad ogni ciclo principale, senza bloccare l'esecuzione generale del sistema.
È utile per controlli ciclici su variabili, ingressi digitali, o per ripetere semplici azioni.

Esempi pratici

Esempio 1 – Stampare un messaggio ciclicamente

```
50 DO 100
100 PRINT "CICLO ATTIVO"
RUN
```

Output atteso:

```
CICLO ATTIVO
CICLO ATTIVO
CICLO ATTIVO
...(continuamente)
```

Esempio 2 – Leggere lo stato di un pulsante

```
10 PINMODE 12 INPUT PULLUP
20 DO 100
100 IF DREAD(12) = 0 THEN PRINT "PREMUTO"
RUN
```

Output atteso:
Stampa "PREMUTO" ogni volta che il pulsante sul pin 5 viene premuto.

Nota:
• È possibile utilizzare più comandi DO per righe differenti
• Non blocca l'esecuzione di altri comandi o servizi come MQTT
• Utile per controlli semplici e ripetitivi
• Valido solo su una singola riga per ogni comando DO

DO BLOCK

Sintassi:
DO BLOCK <inizio> TO <fine>

Descrizione:
Il comando DO BLOCK esegue ciclicamente un blocco di righe del programma, dalla riga iniziale alla riga finale inclusa.
Tutte le righe comprese nel blocco vengono eseguite una volta per ciclo, in ordine.
È utile per raggruppare più istruzioni da eseguire ciclicamente, come controlli, automazioni, reazioni a variabili o messaggi.

Esempi pratici

Esempio 1 – Controllare l'accensione di un LED

```
5 PINMODE 2 OUTPUT NOPULL
10 DO BLOCK 100 TO 110
100 IF TIMEH > 20 THEN DWRITE 2 1
110 IF TIMEH < 21 THEN DWRITE 2 0
RUN
```

Output atteso:
Il LED si accende dopo le 20:00 e si spegne prima delle 21:00, in modo automatico.

Esempio 2 – Reazione a una variabile testuale

```
10 DO BLOCK 100 TO 120
100 IF MSG$ = "ACCENDI" THEN PRINT "LUCE ON"
110 IF MSG$ = "SPEGNI" THEN PRINT "LUCE OFF"
120 LET MSG$ = ""
RUN
```

Output atteso:
Stampa "LUCE ON" o "LUCE OFF" in base al valore della variabile MSG$.

Esempio 3 – Blink LED ogni 500 ms

```
5  PINMODE 2 OUTPUT NOPULL
10 LET S = 0
20 DO BLOCK 100 TO 140
100 IF S = 0 THEN DWRITE 2 1
110 IF S = 1 THEN DWRITE 2 0
120 LET S = S + 1
130 IF S > 1 THEN LET S = 0
140 DELAY 500
RUN
```

Output atteso:
il Led sul pin 2 si accende/spegne ogni mezzo secondo.

Nota:
• Le righe vengono eseguite tutte ad ogni ciclo
• Può contenere IF, LET, DWRITE, WAIT, ecc.
• Non interferisce con MQTT o altre operazioni del sistema
• Si possono usare più blocchi DO BLOCK nel programma

DREAD(p)

Sintassi:
DREAD(pin)

Descrizione:
Legge lo stato digitale del **pin**.

- **Senza DEBOUNCE** (PINMODE senza la parola DEBOUNCE): ritorna il **livello raw** del pin (lettura immediata, può ripetere finché tieni premuto).
- **Con DEBOUNCE** (PINMODE con DEBOUNCE [ms]): ritorna un **impulso one-shot** alla **pressione**:
  - emette **0** (con PULLUP) **una sola volta** quando premi, poi torna a **1** finché non rilasci e ripremi.
  - con PULLDOWN è l'inverso (premuto=1).

Restituisce:

- 1 se il pin è **ALTO** (HIGH, ~3.3V)
- 0 se il pin è **BASSO** (LOW, ~0V)

Usa prima PINMODE per configurare il pin in **INPUT** (con PULLUP, PULLDOWN o NOPULL).
Se vuoi **sempre** il livello raw (ignorando il debounce), usa **DLEVEL(pin)**.

Esempi pratici

Esempio 1 – Lettura diretta (senza debounce)

```
10 PINMODE 12 INPUT PULLUP
20 V = DREAD(12)
30 PRINT "STATO DEL PIN 12: "; V
RUN
```

Output atteso: 0 oppure 1 in base al segnale presente sul pin.

Esempio 2 – Pulsante con PULLUP e DEBOUNCE (one-shot)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DREAD(12) = 0 THEN PRINT "PULSANTE PREMUTO"
30 GOTO 20
RUN
```

Comportamento: stampa **una sola volta** per ogni pressione completa (press→release), evitando ripetizioni mentre tieni premuto.

Esempio 3 – Lettura continua “raw” con DLEVEL (ignora debounce)

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "PULSANTE PREMUTO" ELSE PRINT "RILASCIATO"
30 DELAY 500
40 GOTO 20
RUN
```

Comportamento: mostra lo **stato istantaneo** del pin (0=premuto, 1=rilasciato con PULLUP), a prescindere dal debounce.

Note

- Con **PULLUP**: circuito “attivo-basso” → **premuto=0**, **rilasciato=1**.
  Con **PULLDOWN** è l’inverso (**premuto=1**).
- DEBOUNCE [ms] nel PINMODE filtra i rimbalzi e trasforma DREAD(pin) in un **rilevatore di pressione singola**.
- Per contatori/menu: usa DREAD(pin) con **DEBOUNCE**; per “hold” o durate di pressione, usa DLEVEL(pin).

DWRITE(p, v)

Sintassi:

DWRITE(pin valore)

Descrizione:
Il comando DWRITE imposta un **pin digitale** dell'ESP32 allo stato:

- **HIGH (1)** → tensione 3.3V
- **LOW (0)** → tensione 0V

È usato per **accendere o spegnere LED, attivare relé, segnali di controllo**, ecc.
Il pin deve essere configurato prima come **OUTPUT** usando PINMODE.

Esempi pratici

Esempio 1 – Accendere e spegnere un LED collegato al pin 2

```
10 PINMODE 2 OUTPUT NOPULL
20 DWRITE 2 1
30 WAIT 1000
40 DWRITE 2 0
RUN
```

Output atteso:
Il LED si accende per 1 secondo, poi si spegne.

Esempio 2 – Lampeggio continuo
→ Fa lampeggiare il LED ogni mezzo secondo:

```
10 PINMODE 2 OUTPUT NOPULL
20 DO BLOCK 20 TO 60
30 DWRITE 2 1
40 WAIT 500
50 DWRITE 2 0
60 WAIT 500
RUN
```

Output atteso:
LED collegato al GPIO2 lampeggia a intervalli regolari.

Esempio 3 – Controllo condizionale
→ Attiva un pin solo se una variabile supera una soglia:

```
10 PINMODE 13 OUTPUT NOPULL
20 INPUT A
30 IF A > 100 THEN DWRITE 13 1 ELSE DWRITE 13 0
RUN
```

Output atteso:
Il pin 13 sarà attivo (HIGH) se A è maggiore di 100.

Nota:

- I valori 1 e HIGH sono equivalenti (idem per 0 e LOW)
- È possibile controllare anche pin di output virtuali o logici in alcuni casi.

DIR

Sintassi:

DIR
DIR SD
DIR SPIFFS

Descrizione:

Il comando **DIR** mostra l'elenco dei file presenti nella memoria del sistema.

- Se eseguito **senza parametri**, visualizza il contenuto della memoria **interna SPIFFS**.
- Se viene specificato l'argomento **SD**, mostra invece i file presenti sulla **scheda SD**.
- In alternativa, è possibile indicare esplicitamente **SPIFFS** per elencare la memoria interna (equivalente a DIR senza argomenti).

Questo comando è utile per visualizzare rapidamente i file disponibili, sia su **SPIFFS** che su **SD**, da caricare, cancellare o rinominare.

Esempi pratici

Esempio 1 – Elencare file nella memoria interna SPIFFS

DIR

Output atteso (esempio):

Directory of SPIFFS:
main.bas
prog1.bas

Esempio 2 – Elencare file sulla scheda SD

DIR SD

Output atteso (esempio):

Directory of SD:
program1.bas
demo.bas

Nota:

- DIR accetta un solo argomento opzionale: **SD** o **SPIFFS**.

- Se nessuna scheda SD è collegata o montata, il comando DIR SD mostrerà un messaggio di errore (“Unable to open SD root.”).

ELSE

Sintassi:

IF condizione THEN istruzione1 ELSE istruzione2

Descrizione:

Il costrutto ELSE è parte della struttura condizionale IF...THEN...ELSE.
Permette di eseguire un'**istruzione alternativa** se la **condizione non è vera**.

Può essere usato con singole istruzioni sulla stessa riga oppure con GOTO, PRINT, LET, INPUT, ecc.
BASIC32 non supporta blocchi multi-linea (IF...ENDIF), quindi l'intera logica va espressa su **una singola riga**.

Esempi pratici

Esempio 1 – Verifica di un numero

→ Mostra messaggio diverso a seconda del valore:

```
10 INPUT A
20 IF A > 0 THEN PRINT "POSITIVO" ELSE PRINT "NEGATIVO O ZERO"
RUN
```

Output atteso (se inserisci 5):

```
POSITIVO
```

Esempio 2 – Scelta tra due azioni

→ Accende un LED se il valore è 1, lo spegne altrimenti:

```
10 PINMODE 2 OUTPUT NOPULL
20 INPUT V
30 IF V = 1 THEN DWRITE 2 1 ELSE DWRITE 2 0
RUN
```

Output atteso:

Il pin GPIO2 sarà acceso se V = 1, altrimenti spento.

Esempio 3 – Uso con LET per assegnazioni diverse

```
10 INPUT T
20 IF T < 20 THEN LET STATO$ = "FREDDO" ELSE LET STATO$ = "CALDO"
30 PRINT "STATO: "; STATO$
RUN
```

Output atteso (es. input 15):

STATO: FREDDO

Esempio 4 – Con GOTO per saltare a righe diverse

```
10 INPUT A
20 IF A < 100 THEN GOTO 100 ELSE GOTO 200
100 PRINT "NUMERO PICCOLO": END
200 PRINT "NUMERO GRANDE"
RUN
```

Output atteso (es. input 50):

NUMERO PICCOLO

EXP(x)

Sintassi:

EXP(x)

Descrizione:
La funzione EXP(x) restituisce il valore di **e elevato alla x**, dove **e ≈ 2.71828** è la base dei logaritmi naturali.
È utile per calcoli matematici avanzati, esponenziali, crescita logistica, e operazioni scientifiche.

Il parametro x può essere positivo, negativo o zero.

Esempi pratici

Esempio 1 – Calcolare e^1

```
10 PRINT "EXP(1) = "; EXP(1)
RUN
```

Output atteso:

```
EXP(1) = 2.71828
```

Esempio 2 – e elevato alla seconda

```
10 X = EXP(2)
20 PRINT "EXP(2) = "; X
RUN
```

Output atteso:

```
EXP(2) = 7.389
```

Esempio 3 – Usare EXP con numeri negativi
→ Calcolo di e^-1:

```
10 PRINT "EXP(-1) = "; EXP(-1)
RUN
```

Output atteso:

```
EXP(-1) = 0.3679
```

Esempio 4 – Comparazione con potenze
→ Verifica che EXP(LOG(x)) = x:

```
10 A = 5
20 PRINT "VALORE ORIGINALE: "; A
30 PRINT "EXP(LOG(A)) = "; EXP(LOG(A))
RUN
```

Output atteso:

```
VALORE ORIGINALE: 5
EXP(LOG(A)) = 5
```

FCLOSE

Sintassi

FCLOSE

Descrizione

Chiude il file attualmente aperto, liberando le risorse.

- Devi chiudere SEMPRE un file dopo l'uso.
- Se nessun file è aperto, non produce errore (no-op).

Esempi pratici

Esempio 1 – Chiudere il file

```
10 OPEN "test.txt" W
20 FPRINT "CIAO"
30 FCLOSE
```

Esempio 2 – Apertura multipla

```
10 OPEN "a.txt" W
20 FPRINT "uno"
30 OPEN "b.txt" W    ' a.txt viene chiuso automaticamente
40 FPRINT "due"
50 FCLOSE
```

Note

- Viene automaticamente invocato da OPEN se già esiste un file aperto.
- Sempre sicuro da chiamare.

FNname(...)

Sintassi:

FNnome(arg1, arg2, ...)

Descrizione:

FNname(...) viene usato per **richiamare una funzione personalizzata** precedentemente definita con DEF FN.
Il nome della funzione deve **iniziare con "FN"** e gli argomenti devono corrispondere per **numero e ordine** a quelli usati nella definizione.

La funzione restituisce un valore calcolato in base all'espressione specificata.

Può essere usato:

- in una PRINT
- in un'assegnazione (LET)
- in condizioni logiche (IF)

Esempi pratici

Esempio 1 – Funzione somma

→ Definizione + chiamata:

```
10 DEF FNADD(X,Y) = X + Y
20 PRINT "SOMMA = "; FNADD(5,3)
RUN
```

Output atteso:

```
SOMMA = 8
```

Esempio 2 – Uso in un'espressione più complessa

```
10 DEF FNDOPPIO(X) = X * 2
20 A = FNDOPPIO(4) + 1
30 PRINT "RISULTATO: "; A
RUN
```

Output atteso:

```
RISULTATO: 9
```

Esempio 3 – Richiamo in un ciclo

→ Calcola e stampa il quadrato di ogni numero da 1 a 5:

```
10 DEF FNSQ(X) = X * X
```

```
20 FOR I = 1 TO 5
30 PRINT I; "^2 = "; FNSQ(I)
40 NEXT I
RUN
```

Output atteso:

```
1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
```

Esempio 4 – Uso in condizione IF

→ Funzione che restituisce il triplo, usata in una condizione:

```
10 DEF FNTRIPLO(X) = X * 3
20 INPUT A
30 IF FNTRIPLO(A) > 20 THEN PRINT "GRANDE" ELSE PRINT "PICCOLO"
RUN
```

Output atteso (con input 8):

```
GRANDE
```

FOR/NEXT

Sintassi:

FOR variabile = inizio TO fine [STEP incremento]

Descrizione:

Il comando FOR crea un **ciclo a contatore** che esegue una o più istruzioni finché la variabile indicata non supera il valore finale (in positivo o negativo, a seconda del STEP).

Ogni FOR deve essere **seguito da un NEXT**, che incrementa (o decrementa) la variabile e decide se ripetere il ciclo o uscire.

È utile per:

- Ripetere blocchi di codice un numero definito di volte
- Scorrere coordinate, contatori, indici o sequenze regolari
- Costruire animazioni o loop temporizzati

_____

Esempi pratici

Esempio 1 – Ciclo base da 1 a 5

```
10 FOR I = 1 TO 5
20 PRINT "VALORE: "; I
30 NEXT I
RUN
```

Output atteso:

```
VALORE: 1
VALORE: 2
VALORE: 3
VALORE: 4
VALORE: 5
```

Esempio 2 – Ciclo con STEP negativo (decrescente)

```
10 FOR N = 10 TO 1 STEP -2
20 PRINT "N: "; N
30 NEXT N
RUN
```

Output atteso:

```
N: 10
N: 8
N: 6
N: 4
N: 2
```

Esempio 3 – Cicli annidati (griglia 2x3)

```
10 FOR R = 1 TO 2
20   FOR C = 1 TO 3
30     PRINT "RIGA: "; R; " COLONNA: "; C
40   NEXT C
50 NEXT R
RUN
```

Output atteso:

```
RIGA: 1 COLONNA: 1
RIGA: 1 COLONNA: 2
RIGA: 1 COLONNA: 3
RIGA: 2 COLONNA: 1
RIGA: 2 COLONNA: 2
RIGA: 2 COLONNA: 3
```

_____

Note:

- La variabile viene **creata o aggiornata** automaticamente all'interno del ciclo
- Il valore di STEP può essere positivo o negativo (default: 1)
- I cicli possono essere **annidati** se usano variabili diverse
- I nomi delle variabili in NEXT devono corrispondere esattamente a quelli nel FOR
- NEXT – chiude un ciclo FOR
- RESETFOR – forza la pulizia del ciclo for e svuota tutti i cicli attivi in caso di GOTO o errori

FORMAT

Sintassi

FORMAT SD
FORMAT SPIFFS

Descrizione

FORMAT cancella tutti i file presenti nella directory radice del file system selezionato.
Per sicurezza, se viene eseguito **senza argomenti**, non fa nulla e mostra un messaggio di conferma.

- FORMAT SD → cancella tutti i file sulla **scheda SD**
- FORMAT SPIFFS → cancella tutti i file nella **memoria interna SPIFFS**

Attenzione: i file vengono **eliminati definitivamente** senza possibilità di recupero.

Esempi pratici

Esempio 1 – Protezione da errore

FORMAT

Output:

Specify FORMAT SD or FORMAT SPIFFS to confirm formatting.
Warning: all files on the selected memory will be deleted!

Esempio 2 – Formattare la scheda SD

FORMAT SD

Output:

FORMAT SD: deleted 12 file(s)

Esempio 3 – Formattare la memoria interna SPIFFS

FORMAT SPIFFS

Output:

FORMAT SPIFFS: deleted 5 file(s)

Nota:

- Se la SD non è inserita o non accessibile, FORMAT SD mostrerà IO ERROR: SD root not accessible.

FPRINT

Sintassi

FPRINT <espressione_stringa>

Descrizione

Scrive una **riga** nel file aperto tramite OPEN.

- Aggiunge automaticamente un carattere **newline** a fine riga.
- Riceve una qualunque espressione che restituisce una stringa:
  - "CIAO"
  - STR$(A)
  - "TEMP=" + STR$(T)
- Può essere usato solo se il file è aperto in modalità **W** o **A**.

Esempi pratici

Esempio 1 – Riga semplice

```
10 OPEN "test.txt" W
20 FPRINT "CIAO MONDO"
30 FCLOSE
```

Esempio 2 – Riga con numeri

```
10 TEMP = 24.6
20 OPEN "sensore.txt" A
30 FPRINT "TEMP=" + STR$(TEMP)
40 FCLOSE
```

Note

- In caso di file non aperto o in modalità non valida → errore.
- FPRINT non stampa sul terminale → scrive SOLO nel file.

FREAD$

Sintassi

FREAD$

Descrizione

Legge **una riga** dal file aperto con OPEN.

- Ritorna una **stringa** (senza newline finale).
- Ritorna "" se si raggiunge l'**EOF** (fine file).
- Può essere usato:
  - in un assegnamento
  - direttamente in PRINT
  - in condizioni IF

Esempi pratici

Esempio 1 – Leggere tutte le righe

```
10 OPEN "log.txt" R
20 RIGA$ = FREAD$
30 IF RIGA$ = "" THEN FCLOSE : END
40 PRINT RIGA$
50 GOTO 20
```

Esempio 2 – Lettura singola

```
10 OPEN SD "config.txt" R
20 PRINT "Prima riga=", FREAD$
30 FCLOSE
```

Note

- Funziona solo in modalità **R** e **A**.
- Ogni chiamata legge la riga successiva.
- Non includere il newline (\n), che viene automaticamente rimosso.

FREEMEM

Sintassi:
PRINT FREEMEM

Descrizione:
Il comando FREEMEM restituisce la quantità di memoria libera disponibile in byte per l'esecuzione del programma BASIC.
È utile per monitorare l'utilizzo della RAM e prevenire problemi legati a esaurimento di memoria.

Esempi pratici

Esempio 1 – Visualizzare la memoria libera

```
10 PRINT FREEMEM
RUN
```

Output atteso:

```
22480
```

Nota:
• Il valore restituito è in byte
• Non richiede parametri
• Utile per debugging e diagnostica
• Valido solo come espressione in PRINT

FSDEFAULT

Sintassi:

FSDEFAULT SD
FSDEFAULT SPIFFS

Descrizione:

Imposta il filesystem predefinito utilizzato dai comandi che accedono ai file.
Dopo aver impostato il valore, tutti i comandi che non specificano esplicitamente **SD** o **SPIFFS** useranno automaticamente il filesystem scelto.

Il valore viene memorizzato nelle preferenze (/prefs.cfg) ed è mantenuto anche dopo il riavvio dell'ESP32.

Effetti del filesystem predefinito sui comandi BASIC:

Puoi sempre forzare manualmente un filesystem scrivendo SD o SPIFFS dopo il comando (es: DIR SD, SAVE SPIFFS "FILE").

Esempi pratici

Esempio 1 – Impostare SD come archivio predefinito

```
10 FSDEFAULT SD
20 SAVE "TEST"
RUN
```

Effetto:
Il file **TEST.BAS** viene salvato su SD anche senza scrivere SAVE SD.

Esempio 2 – Tornare a SPIFFS come archivio predefinito

```
10 FSDEFAULT SPIFFS
20 DIR
RUN
```

Effetto:
DIR mostra il contenuto dello SPIFFS perché ora è il filesystem di default.

Esempio 3 – Default SD, ma un comando forzato su SPIFFS

```
10 FSDEFAULT SD
```

```
20 SAVE SPIFFS "CONFIG"
RUN
```

Effetto:
Anche se il default è SD, il file **CONFIG.BAS** viene salvato nello SPIFFS perché specificato esplicitamente.

Note:

- Il valore predefinito originale è **SPIFFS**.
- Se il default è **SD**, ma la SD non è montata, il comando produce un messaggio di avviso.
- Il comando **FSDEFAULT** non richiede parametri numerici: accetta solo SD o SPIFFS.
- Le preferenze vengono salvate automaticamente dopo il comando.

FUNC / ENDFUNC

Sintassi:

FUNC <nome>
FUNC <nome> LOOP
...
ENDFUNC

Descrizione:

Il comando FUNC definisce una funzione utente. Le funzioni permettono di creare blocchi riutilizzabili di codice.
Quando è presente la parola chiave LOOP, la funzione viene eseguita ciclicamente in background (non bloccante) tramite STARTFUNC.
Tutte le funzioni devono essere chiuse da ENDFUNC.

Esempi pratici

Esempio 1 – Funzione semplice (non in loop)

```
5 FUNC SALUTO
10 PRINT "Ciao dal Basic!"
20 ENDFUNC
30 CALLFUNC SALUTO
```

Output atteso:

Ciao dal Basic!

Esempio 2 – Funzione ciclica (loop) per lampeggio LED

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC BLINK LOOP
20 DWRITE 2 1
30 DELAY 300
40 DWRITE 2 0
50 DELAY 300
60 ENDFUNC
70 STARTFUNC BLINK
```

Output atteso:
Il LED lampeggia ogni 300 ms in background.

Note:

- Ogni funzione deve iniziare con FUNC nome e finire con ENDFUNC
- Il nome deve essere una parola unica (senza spazi o simboli)

- Il codice all'interno non viene eseguito da solo: va richiamato con CALLFUNC o STARTFUNC
- Se definita con LOOP, la funzione gira in modo continuo e parallelo
- Le funzioni cicliche vanno interrotte con STOPFUNC
- Può contenere qualsiasi comando BASIC (eccetto FUNC, ENDFUNC annidati)

...TO...STEP...NEXT

Sintassi:

```
FOR variabile = inizio TO fine [STEP incremento]
  ...
NEXT [variabile]
```

Descrizione:
La struttura FOR...NEXT viene utilizzata per creare **cicli con contatore**, in cui una variabile numerica viene **incrementata o decrementata automaticamente** fino a raggiungere un valore finale.

- variabile: nome del contatore (es. I)
- inizio: valore iniziale
- fine: valore finale
- STEP: incremento (positivo o negativo, opzionale — predefinito = 1)

Il corpo del ciclo può contenere qualsiasi istruzione, inclusi IF, PRINT, LET, GOTO, ecc.

Esempi pratici

Esempio 1 – Ciclo da 1 a 5 con incremento di 1

```
10 FOR I = 1 TO 5
20 PRINT "VALORE: "; I
30 NEXT I
RUN
```

Output atteso:

```
VALORE: 1
VALORE: 2
VALORE: 3
VALORE: 4
VALORE: 5
```

Esempio 2 – Ciclo con incremento personalizzato (STEP 2)

```
10 FOR N = 0 TO 10 STEP 2
20 PRINT "N = "; N
30 NEXT N
RUN
```

Output atteso:

```
N = 0
N = 2
N = 4
N = 6
```

```
N = 8
N = 10
```

Esempio 3 – Ciclo decrescente (STEP negativo)

```
10 FOR X = 10 TO 1 STEP -3
20 PRINT X
30 NEXT X
RUN
```

Output atteso:

```
10
7
4
1
```

Esempio 4 – Calcolare la somma dei numeri da 1 a 10

```
10 SUM = 0
20 FOR I = 1 TO 10
30 SUM = SUM + I
40 NEXT I
50 PRINT "SOMMA = "; SUM
RUN
```

Output atteso:

```
SOMMA = 55
```

GET

Sintassi:

GET

Descrizione:
Il comando GET legge **un singolo carattere dalla tastiera** (terminale seriale) **senza attendere il tasto INVIO**.
Restituisce il **codice ASCII** del carattere premuto. Se non viene premuto nulla, può restituire -1 o restare in attesa (a seconda del firmware).

È utile per:

- leggere tasti **in tempo reale**
- costruire **interfacce interattive**
- leggere **sequenze di comandi** o input manuali

Esempi pratici

Esempio 1 – Premere un tasto e visualizzarne il codice ASCII

```
10 PRINT "PREMI UN TASTO:"
20 A = GET
30 PRINT "CODICE ASCII: "; A
RUN
```

Output atteso (se premi ad esempio la lettera A):

```
PREMI UN TASTO:
CODICE ASCII: 65
```

Esempio 2 – Leggere più tasti in un ciclo
→ Continua a leggere finché non premi Q (ASCII 81)

```
10 C = GET
20 IF C <> -1 THEN PRINT "TASTO: "; C
30 IF C = 81 THEN PRINT "fine" : END
40 GOTO 10
RUN
```

Output atteso:

```
TASTO: 72
TASTO: 69
TASTO: 76
TASTO: 76
TASTO: 79
TASTO: 81
```

Esempio 3 – Eseguire azioni in base al tasto premuto
→ Accende un LED con 1, lo spegne con 0

```
10 PINMODE 2 OUTPUT NOPULL
20 PRINT "PREMI 1 PER ON, 0 PER OFF"
30 DO BLOCK 40 TO 60
40 C = GET
50 IF C = 49 THEN DWRITE 2 1
60 IF C = 48 THEN DWRITE 2 0
RUN
```

Output atteso:

- Se premi 1, il LED si accende
- Se premi 0, il LED si spegne

Nota:

- GET può restituire -1 se non ci sono caratteri in coda
- I codici ASCII di tasti comuni:
  0 → 48, 1 → 49, A → 65, a → 97

GOSUB n

Sintassi:

GOSUB numero_riga

Descrizione:
Il comando GOSUB consente di **saltare a una subroutine** (blocco di codice) definita a un'altra riga del programma, ed eseguirla.
Al termine della subroutine, si usa RETURN per **tornare alla riga successiva a quella da cui è stato chiamato GOSUB**.

È utile per:

- **riutilizzare codice**
- **strutturare** il programma in **blocchi logici**
- creare funzioni operative senza DEF FN

Puoi usare più GOSUB e annidarli, ma ogni GOSUB deve avere un corrispondente RETURN.

Esempi pratici

Esempio 1 – Subroutine che stampa una riga

```
10 GOSUB 100
20 PRINT "PROGRAMMA PRINCIPALE"
30 END
100 PRINT "SUBROUTINE ESEGUITA"
110 RETURN
RUN
```

Output atteso:

```
SUBROUTINE ESEGUITA
PROGRAMMA PRINCIPALE
```

Esempio 2 – Chiamare la stessa subroutine più volte

```
10 FOR I = 1 TO 3
20 GOSUB 100
30 NEXT I
40 END
100 PRINT "ESECUZIONE NUMERO: "; I
110 RETURN
RUN
```

Output atteso:

```
ESECUZIONE NUMERO: 1
ESECUZIONE NUMERO: 2
```

ESECUZIONE NUMERO: 3

Esempio 3 – Subroutine per calcolo riutilizzabile

```
10 INPUT A, B
20 GOSUB 100
30 PRINT "RISULTATO: "; R
40 END
100 R = A * B
110 RETURN
RUN
```

Output atteso (es. input 3, 4):

RISULTATO: 12

Esempio 4 – Errore comune da evitare

→ Se dimentichi RETURN, il programma **non torna** indietro correttamente.

```
10 GOSUB 100
20 PRINT "QUESTA NON VERRÀ MAI ESEGUITA"
100 PRINT "DIMENTICATO IL RETURN"
```

Corretto invece con:

```
10 GOSUB 100
20 PRINT "QUESTA VERRÀ VISUALIZZATA"
30 END
100 PRINT "CON RETURN"
110 RETURN
```

GOTO n

Sintassi:

GOTO numero_riga

Descrizione:
Il comando GOTO trasferisce l'esecuzione del programma alla **riga con numero n**.
È uno strumento basilare ma potente per **saltare blocchi di codice**, **creare loop manuali**, o **diramare il flusso del programma** in base a condizioni.

Tuttavia, è consigliabile usarlo con criterio per mantenere il codice leggibile (evita il cosiddetto “spaghetti code”).

Esempi pratici

Esempio 1 – Salto semplice
→ Salta una riga del programma:

```
10 PRINT "INIZIO"
20 GOTO 40
30 PRINT "QUESTA NON SI VEDE"
40 PRINT "DOPO IL SALTO"
RUN
```

Output atteso:

```
INIZIO
DOPO IL SALTO
```

Esempio 2 – Creazione di un ciclo manuale
→ Stampa i numeri da 1 a 5 senza usare FOR:

```
10 A = 1
20 PRINT A
30 A = A + 1
40 IF A <= 5 THEN GOTO 20
RUN
```

Output atteso:

```
1
2
3
4
5
```

Esempio 3 – Gestione di menù testuale
→ Semplice menù con salti condizionati:

```
10 PRINT "1. START"
20 PRINT "2. ESCI"
30 INPUT C
40 IF C = 1 THEN GOTO 100
50 IF C = 2 THEN GOTO 200
100 PRINT "INIZIO GIOCO": END
200 PRINT "USCITA DAL PROGRAMMA"
RUN
```

Output atteso (se premi 1):

```
INIZIO GIOCO
```

Esempio 4 – Evitare loop infiniti involontari
→ Usa una condizione per controllare il ciclo:

```
10 PRINT "CICLO CONTINUO"
20 GOTO 10
```

Questo ciclo è **infinito** e va interrotto manualmente.

Nota:

- Le righe di destinazione devono esistere, altrimenti il programma può bloccarsi.
- GOTO può essere usato dentro IF, ELSE, cicli o subroutine.

HTMLOBJ

Sintassi:

HTMLOBJ "<riga_html>"
HTMLOBJ espressione_stringa$

Descrizione:

Il comando **HTMLOBJ** memorizza una riga di codice HTML da includere nella pagina web generata dal dispositivo.

Ogni chiamata a HTMLOBJ accoda la stringa alla pagina HTML interna.
Le righe accumulate vengono inviate al browser quando viene eseguito **HTMLSTART**.

L'argomento può essere:

- una stringa costante racchiusa tra virgolette
- una variabile stringa (es. A$)
- una espressione stringa (es. "Valore: " + STR$(T))

Le righe HTML restano memorizzate finché non viene eseguito MEMCLEAN(HTML) o si riavvia la scheda.

Esempi pratici

Esempio 1 – Intestazione HTML semplice

```
10 WIFI "SSID" "PASSWORD"
20 HTMLOBJ "<h1>Benvenuto su Basic32</h1>"
30 HTMLSTART
```

⟶ Mostra un titolo grande nella pagina web.

Esempio 2 – Mostrare una variabile BASIC

```
10 WIFI "SSID" "PASSWORD"
20 LET T = 23.7
30 LET R$ = "<p>Temperatura attuale: " + STR$(T) + " &deg;C</p>"
40 HTMLOBJ "<h2>Sensore</h2>"
50 HTMLOBJ R$
60 HTMLSTART
```

⟶ Inserisce nella pagina il valore di una variabile BASIC formattata come HTML.

Esempio 3 – Tabella HTML con valori

```
10 WIFI "SSID" "PASSWORD"
20 LET T = 22.5
30 LET H = 55.0
40 HTMLOBJ "<h2>Lettura sensori</h2>"
50 HTMLOBJ "<table border='1' style='margin:auto;border-collapse:collapse;'>"
60 HTMLOBJ "<tr><th style='padding:6px;'>Grandezza</th><th
style='padding:6px;'>Valore</th></tr>"
70 LET RIGA$ = "<tr><td style='padding:6px;'>Temperatura</td><td
style='padding:6px;'>" + STR$(T) + " &deg;C</td></tr>"
80 HTMLOBJ RIGA$
90 LET RIGA$ = "<tr><td style='padding:6px;'>Umidit&agrave;</td><td
style='padding:6px;'>" + STR$(H) + " %</td></tr>"
100 HTMLOBJ RIGA$
110 HTMLOBJ "</table>"
120 HTMLSTART
```

⟶ Costruisce una tabella HTML usando stringhe generate dal BASIC.

Esempio 4 – Pulsanti per controllare un LED senza cambiare pagina

```
10 PINMODE 32 OUTPUT NOPULL
20 WIFI "ssid" "password"
30 HTMLOBJ "<div style='font-family:Arial,sans-serif;background:#202124;color:#fff;text-
align:center;min-height:100vh;display:flex;flex-direction:column;align-items:center;justify-
content:center;'>"
40 HTMLOBJ "<h2 style='font-size:32px;margin-bottom:40px;'>Controllo LED on
board</h2>"
50 HTMLOBJ "<iframe name='exec_hidden'
style='display:none;width:0;height:0;border:0;'></iframe>"
60 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%201' target='exec_hidden'>"
70 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-
radius:10px;border:none;cursor:pointer;background:#4caf50;color:#fff;'>Accendi
LED</button>"
80 HTMLOBJ "</a>"
90 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%200' target='exec_hidden'>"
100 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-
radius:10px;border:none;cursor:pointer;background:#f44336;color:#fff;'>Spegni
LED</button>"
110 HTMLOBJ "</a>"
120 HTMLOBJ "</div>"
130 HTMLSTART
```

⟶ I pulsanti inviano comandi a /exec ma la pagina rimane sempre la stessa (la risposta “OK” viene caricata nell’iframe nascosto).

Output atteso:

Le righe HTML passate a HTMLOBJ vengono concatenate e costituiranno il contenuto della pagina web servita dopo HTMLSTART, nell'ordine in cui sono state definite.

Nota:

- Le stringhe costanti devono essere racchiuse tra virgolette doppie.
- Per inserire doppi apici nell'HTML è consigliato usare apici singoli ' oppure raddoppiare i doppi apici:
  class=""bottone""
- HTMLOBJ non valida l'HTML: eventuali errori di markup si rifletteranno nella pagina.
- Le righe HTML vengono accumulate tra diverse esecuzioni: per ripartire da zero usare:
- MEMCLEAN(HTML)
- HTMLOBJ non avvia nessun server web: per rendere visibile la pagina è necessario chiamare HTMLSTART.

HTMLSTART

Sintassi:

HTMLSTART

Descrizione:

Il comando **HTMLSTART** avvia il server web integrato e rende accessibile via browser la pagina HTML costruita in precedenza tramite **HTMLOBJ**.

Il dispositivo deve essere già connesso in Wi-Fi, tramite:

- WIFI "ssid" "password" (modalità stazione)
- WIFIAP "ssid" "password" (modalità Access Point, se supportato)

La pagina generata è raggiungibile all'indirizzo IP riportato dai comandi:

- IP → indirizzo in modalità Wi-Fi client
- IPAP → indirizzo in modalità Access Point

Esempi pratici

Esempio 1 – Pagina di test

```
10 WIFI "ssid" "password"
20 HTMLOBJ "<h2>Pagina di test Basic32</h2>"
30 HTMLOBJ "<p>Se vedi questo messaggio, HTMLSTART funziona!</p>"
40 HTMLSTART
```

⟶ Avvia una pagina molto semplice per verificare il funzionamento del web server.

Esempio 2 – Controllo LED via Web (senza cambiare pagina)

```
10 PINMODE 32 OUTPUT NOPULL
20 WIFI "ssid" "password"
30 HTMLOBJ "<div style='font-family:Arial,sans-serif;background:#202124;color:#fff;text-
align:center;min-height:100vh;display:flex;flex-direction:column;align-items:center;justify-
content:center;'>"
40 HTMLOBJ "<h2 style='font-size:32px;margin-bottom:40px;'>Controllo LED on
board</h2>"
50 HTMLOBJ "<iframe name='exec_hidden'
style='display:none;width:0;height:0;border:0;'></iframe>"
60 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%201' target='exec_hidden'>"
70 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-
radius:10px;border:none;cursor:pointer;background:#4caf50;color:#fff;'>Accendi
LED</button>"
```

```
80 HTMLOBJ "</a>"
90 HTMLOBJ "<a href='/exec?cmd=DWRITE%2032%200' target='exec_hidden'>"
100 HTMLOBJ "<button style='font-size:24px;padding:12px 24px;margin:10px;border-
radius:10px;border:none;cursor:pointer;background:#f44336;color:#fff;'>Spegni
LED</button>"
110 HTMLOBJ "</a>"
120 HTMLOBJ "</div>"
130 HTMLSTART
```

⟶ Permette di controllare il LED dal browser. I comandi vengono eseguiti su /exec e la pagina dei pulsanti non viene sostituita dalla scritta “OK”.

Output atteso:

Dopo l’esecuzione di HTMLSTART:

- Il server web è attivo sull’IP del dispositivo.
- La pagina HTML creata con HTMLOBJ è visibile e navigabile da browser.
- Gli eventuali link o pulsanti che puntano a /exec?cmd=... eseguono comandi BASIC sul dispositivo.

Nota:

- Deve essere chiamato **dopo** la configurazione Wi-Fi (WIFI / WIFIAP).
- Richiede che siano state definite una o più righe HTMLOBJ, altrimenti la pagina risulterà vuota o minimale.
- Per evitare che vecchie pagine si mescolino alle nuove è consigliato usare MEMCLEAN(HTML) all’inizio dei programmi che definiscono contenuti HTML.
- Le azioni lato BASIC invocate via web passano dall’endpoint:
- /exec?cmd=<comando BASIC>
- È possibile usare <iframe> (come negli esempi) o JavaScript per eseguire comandi senza cambiare pagina principale.

HTTP GET

Sintassi

HTTP GET "url" TO var$

Descrizione

Esegue una richiesta **HTTP GET** (solo http://) e salva **il corpo della risposta** nella variabile indicata.

- Se l'URL inizia con https:// → **errore: "HTTPS NON COMPATIBILE".**
- La variabile speciale **HTTPSTATUS** contiene il codice HTTP (200=OK).
- Gli header completi possono essere letti con:
  HTTP HEADERS TO var$.

Esempio 1 – IP pubblico

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://ip-api.com/json" TO R$
30 IF HTTPSTATUS<>200 THEN PRINT "Errore HTTP ";HTTPSTATUS : GOTO 90
40 HTTP JSON GET "query" FROM R$ TO I$
50 PRINT "IP pubblico=";I$
90 END
```

Esempio 2 – Meteo OpenWeatherMap

```
10 WIFI "SSID" "PASSWORD"
20 LET
URL$="http://api.openweathermap.org/data/2.5/weather?q=Milano,it&units=metric&appid=
LA_TUA_API_KEY"
30 HTTP GET URL$ TO R$
40 IF HTTPSTATUS<>200 THEN PRINT "Errore HTTP ";HTTPSTATUS : GOTO 100
50 HTTP JSON GET "main.temp" FROM R$ TO T$
60 HTTP JSON GET "weather[0].description" FROM R$ TO D$
70 PRINT "Meteo:";T$;"°C ";D$
100 END
```

Note

- Richiede Wi-Fi attivo (WIFI "ssid" "pwd").
- L'header **Accept-Encoding: identity** è forzato → risposta sempre in **testo non compresso**.
- Se la risposta non è JSON, puoi stamparla direttamente.

HTTP HEADERS

Sintassi

HTTP HEADERS TO var$

Descrizione

Salva **gli header HTTP completi** dell'ultima risposta GET/POST nella variabile indicata.

Esempio pratico

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://ip-api.com/json" TO R$
30 HTTP HEADERS TO H$
40 PRINT "Status=";HTTPSTATUS
50 PRINT LEFT$(H$,200)
60 PRINT LEFT$(R$,200)
90 END
```

Note

- Restituisce **gli header grezzi** in un'unica stringa.
- Utile per diagnosticare redirect, Content-Type, Content-Length.

HTTP JSON GET

Sintassi

HTTP JSON GET path$ FROM src$ TO var$

Descrizione

Estrae un valore da una stringa contenente **una risposta JSON**.

- path$ può includere:
  - campi annidati → main.temp
  - indici array → weather[0].description
- Se il percorso non esiste → **errore: JSON Path not found.**
- La funzione è **robusta**:
  - ignora tutto ciò che precede il primo { o [
    (quindi funziona anche se nel buffer ci sono header)

Esempio 1 – Meteo OpenWeatherMap

```
10 WIFI "SSID" "PASSWORD"
20 LET
URL$="http://api.openweathermap.org/data/2.5/weather?q=Milano,it&units=metric&appid=
LA_TUA_API_KEY"
30 HTTP GET URL$ TO R$
40 HTTP JSON GET "main.temp" FROM R$ TO T$
50 HTTP JSON GET "weather[0].description" FROM R$ TO D$
60 PRINT "Temp=";T$;"°C ";D$
90 END
```

Esempio 2 – Host da httpbin

```
10 WIFI "SSID" "PASSWORD"
20 HTTP GET "http://httpbin.org/get" TO R$
30 HTTP JSON GET "headers.Host" FROM R$ TO H$
40 PRINT "Server visto come host:";H$
90 END
```

Note

- La funzione cerca automaticamente il **primo { o [** per identificare l'inizio del JSON.
- Gli indici array partono da 0.
- Controllare sempre HTTPSTATUS=200 prima di estrarre.

HTTP POST

Sintassi

HTTP POST "url" "contentType" "body" TO var$

Descrizione

Invia una richiesta **HTTP POST** (solo http://) con corpo e Content-Type specificati.

- HTTPS → **non supportato**.
- La risposta viene salvata in var$.
- Stato HTTP disponibile in HTTPSTATUS.

IMPORTANTE
Se il corpo contiene virgolette ", va costruito con CHR$(34)
oppure verrà troncato.

Esempio 1 – Echo JSON (corretto e funzionante)

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://httpbin.org/post"
30 LET CT$="application/json"
40 LET Q$ = CHR$(34)
50 LET B$ = "{" + Q$ + "hello" + Q$ + ":" + Q$ + "world" + Q$ + "}"
60 HTTP POST URL$ CT$ B$ TO R$
70 HTTP JSON GET "json.hello" FROM R$ TO V$
80 PRINT "Server ha ricevuto:";V$
90 END
```

Output corretto:

Server ha ricevuto:world

Esempio 2 – Invia un sensore

```
10 WIFI "SSID" "PASSWORD"
20 LET URL$="http://mioserver.local/api/save"
30 LET CT$="application/x-www-form-urlencoded"
40 LET B$="name=temp&value=23.7"
50 HTTP POST URL$ CT$ B$ TO R$
60 PRINT "Status=";HTTPSTATUS
70 PRINT LEFT$(R$,120)
90 END
```

Note

- Se il server restituisce JSON → puoi usare HTTP JSON GET.
- Evitare body troppo grandi (limiti di memoria dell’ESP32).
- Funziona solo con server HTTP (porta 80, testo non compresso).

IF ... THEN [ELSE]

Sintassi:

IF condizione THEN istruzione [ELSE istruzione]

Descrizione:

IF ... THEN è il costrutto condizionale principale di BASIC32.
Valuta una **condizione logica** e, se vera, esegue l'istruzione indicata dopo THEN.
Se la condizione è falsa e viene specificato ELSE, esegue l'istruzione alternativa.

- Supporta operatori: =, <>, <, <=, >, >=
- È compatibile con numeri, stringhe e funzioni
- Le istruzioni devono stare **sulla stessa riga**

Esempi pratici

Esempio 1 – Condizione semplice

→ Controlla se un numero è maggiore di 10:

```
10 INPUT A
20 IF A > 10 THEN PRINT "MAGGIORE DI 10"
RUN
```

Output atteso (se inserisci 15):

```
MAGGIORE DI 10
```

Esempio 2 – Condizione con ELSE

→ Mostra due messaggi alternativi:

```
10 INPUT A
20 IF A = 0 THEN PRINT "ZERO" ELSE PRINT "NON ZERO"
RUN
```

Output atteso (se inserisci 0):

```
ZERO
```

Esempio 3 – Uso con GOTO

→ Diramazione del flusso in base a una scelta:

```
10 INPUT S
20 IF S = 1 THEN GOTO 100 ELSE GOTO 200
100 PRINT "SCELTA 1": END
200 PRINT "ALTRA SCELTA"
RUN
```

Output atteso (con input 2):

ALTRA SCELTA

Esempio 4 – Condizione su stringhe

→ Confronta due stringhe:

```
10 INPUT A$
20 IF A$ = "CIAO" THEN PRINT "SALUTO RICONOSCIUTO" ELSE PRINT "STRINGA DIVERSA"
RUN
```

Output atteso (con input CIAO):

SALUTO RICONOSCIUTO

Esempio 5 – Condizione negativa

→ Usa <> per “diverso da”:

```
10 INPUT X
20 IF X <> 0 THEN PRINT "DIVERSO DA ZERO"
RUN
```

Nota:

- Le condizioni devono restituire **vero/falso** (valori numerici logici)
- Solo una **singola istruzione** può seguire THEN e ELSE sulla riga

ILI CIRCLE

Sintassi:
ILI CIRCLE <x> <y> <r> [r g b] [drawnow]

Descrizione:
Disegna un cerchio vuoto (solo il contorno).

- <x> <y>: centro del cerchio.
- <r>: raggio in pixel.
- [r g b]: colore opzionale (default bianco).
- [drawnow]: opzionale (1 = disegna subito, 0 = accoda).

Esempi pratici

Esempio 1 – Cerchio rosso immediato
10 ILI CIRCLE 60 120 30 255 0 0
Disegna subito un cerchio vuoto rosso di raggio 30 centrato in (60,120).

Esempio 2 – Cerchio magenta differito
10 ILI CIRCLE 80 140 20 255 0 255 0
20 ILI DRAW
Il cerchio appare solo dopo ILI DRAW.

Note
• Per un cerchio pieno, usa ILI FILLCIRCLE.
• Utile per pulsanti, contorni o indicatori.

ILI CLEAR

Sintassi:

ILI CLEAR

Descrizione:

Cancella lo schermo riempiendolo con il colore di sfondo corrente. Il colore di sfondo può essere impostato con ILI SETBGCOLOR.

Non rimuove sprite o dati memorizzati, ma solo ciò che è graficamente visibile.

Esempi pratici

Esempio 1 – Pulizia dello schermo in blu

```
10 ILI SETBGCOLOR 0 0 255
20 ILI CLEAR
```

Risultato: lo schermo viene completamente riempito di blu.

Esempio 2 – Dopo il disegno

```
10 ILI RECT 10 10 100 50 255 0 0
20 ILI CLEAR
```

Risultato: il rettangolo rosso viene cancellato e lo schermo torna al colore di sfondo.

Note

- Non rimuove sprite memorizzati.
- Per rimuovere completamente sprite dallo schermo, usare ILI SPRITE CLEAR

ILI DRAW

Sintassi:
ILI DRAW

Descrizione:
Esegue tutti i comandi grafici che sono stati chiamati con [drawnow] = 0.
Vengono disegnati nell'ordine di inserimento:
PIXEL, LINE, RECT, FILLRECT, CIRCLE, FILLCIRCLE, TEXT.
Dopo il disegno, la coda viene svuotata.

Esempi pratici

Esempio 1 – Disegno immediato
10 ILI LINE 10 10 100 50 255 0 0
La linea rossa è visibile subito.

Esempio 2 – Disegno bufferizzato
10 ILI RECT 20 20 100 50 255 0 0 0
20 ILI FILLRECT 130 50 60 40 255 255 0 0
30 ILI TEXT 10 200 2 "CIAO BUFFER" 255 255 255 0
40 ILI DRAW
Tutti i comandi vengono disegnati insieme alla riga 40.

Note
• ILI DRAW mostra contemporaneamente tutto ciò che è stato accodato con drawnow=0.
• Utile per evitare sfarfallii o per aggiornamenti di schermata completi (frame buffer simulato).

ILI FILLCIRCLE

Sintassi:
ILI FILLCIRCLE <x> <y> <r> [r g b] [drawnow]

Descrizione:
Disegna un cerchio pieno di raggio <r> centrato in (x, y).

- [r g b]: colore del riempimento (default bianco).
- [drawnow]: opzionale (1 = disegna subito, 0 = accoda).

Esempi pratici

Esempio 1 – Cerchio pieno rosso immediato
10 ILI FILLCIRCLE 100 100 30 255 0 0
Disegna subito un cerchio pieno rosso di raggio 30 centrato in (100,100).

Esempio 2 – Cerchio pieno ciano differito
10 ILI FILLCIRCLE 140 140 20 0 255 255 0
20 ILI DRAW
Il cerchio viene disegnato solo alla riga 20.

Note
• Se [r g b] non è specificato, il colore è bianco.
• Il parametro [drawnow] permette di accodare il cerchio per disegnarlo insieme ad altri elementi.

ILI FILLRECT

Sintassi:
ILI FILLRECT <x> <y> <w> <h> [r g b] [drawnow]

Descrizione:
Disegna un rettangolo pieno alle coordinate specificate.

- <x> <y>: coordinate dell'angolo superiore sinistro.
- <w> <h>: larghezza e altezza del rettangolo.
- [r g b]: colore del riempimento (default bianco).
- [drawnow]: opzionale (1 = disegna subito, 0 = accoda).

Esempi pratici

Esempio 1 – Rettangolo pieno bianco immediato
10 ILI FILLRECT 10 10 50 20
Disegna subito un rettangolo pieno bianco di 50x20 pixel a (10,10).

Esempio 2 – Rettangolo verde differito
10 ILI FILLRECT 40 80 60 30 0 255 0 0
20 ILI DRAW
Il rettangolo verde viene disegnato solo alla riga 20.

Note
• Usa ILI FILLRECT per sfondi, pulsanti o aree colorate.
• Combinato con [drawnow]=0 puoi comporre intere schermate e visualizzarle in un solo frame.

ILI INIT (ILI9341)

Sintassi

ILI INIT cs dc rst rot
ILI INIT cs dc rst rot [miso] [mosi] [sck]

Descrizione

Inizializza il display TFT ILI9341, specificando i pin di connessione e l'orientamento dello schermo.

Parametri:

- **cs**: pin Chip Select
- **dc**: pin Data/Command
- **rst**: pin Reset
- **rot**: rotazione dello schermo (valori ammessi: 0, 1, 2, 3)
- **miso** *(opzionale)*: pin MISO del bus SPI
- **mosi** *(opzionale)*: pin MOSI del bus SPI
- **sck** *(opzionale)*: pin SCK/CLK del bus SPI

Se i pin SPI opzionali **non vengono specificati**, viene utilizzata la configurazione SPI di default già presente nel sistema.
Se invece vengono specificati, il bus SPI viene inizializzato con i pin indicati.

Una volta inizializzato, il display è pronto per ricevere comandi grafici (linee, rettangoli, testo, sprite, ecc.).

Esempi pratici

Esempio 1 – Inizializzazione con rotazione 0

10 ILI INIT 17 16 5 0

Risultato: lo schermo viene inizializzato con i pin specificati e orientamento normale (orizzontale).

Esempio 2 – Rotazione verticale (valore 1)

10 ILI INIT 17 16 5 1

Risultato: lo schermo viene ruotato di 90° (utile per visualizzazioni verticali).

Esempio 3 – Inizializzazione con pin SPI personalizzati

10 ILI INIT 17 16 5 0 19 23 18

Risultato: il display viene inizializzato usando i pin SPI indicati (MISO=19, MOSI=23, SCK=18).

Note

- Questo comando deve essere eseguito prima di qualsiasi altro comando ILI.
- La rotazione può essere modificata anche in seguito, rieseguendo ILI INIT con un nuovo valore.
- Dopo l’inizializzazione, lo schermo viene automaticamente pulito (riempito di nero).
- Si possono avere sul display al massimo **256 sprite** e **512 elementi**.

ILI INIT TOUCH (XPT2046)

Sintassi

ILI INIT TOUCH cs rotation
ILI INIT TOUCH cs rotation [miso] [mosi] [sck]
ILI INIT TOUCH cs rotation [miso] [mosi] [sck] [irq]

Descrizione

Il comando ILI INIT TOUCH inizializza il controller touch XPT2046 collegato al display ILI9341.

Parametri:

- **cs**: pin Chip Select del controller touch
- **rotation**: definisce l'orientamento del display (0–3) ed è sincronizzato con la rotazione del TFT
- **miso** *(opzionale)*: pin MISO del bus SPI
- **mosi** *(opzionale)*: pin MOSI del bus SPI
- **sck** *(opzionale)*: pin SCK/CLK del bus SPI
- **irq** *(opzionale)*: pin IRQ/INT del touch (solo se fisicamente collegato)

Comportamento:

- Se i pin SPI **non vengono specificati**, viene usata la configurazione SPI di default.
- Se vengono specificati miso mosi sck, lo SPI viene inizializzato con quei pin.
- Se **irq non viene specificato**, il sistema utilizza automaticamente la modalità **senza IRQ** (polling SPI).
- Se irq viene specificato, viene utilizzato come linea di interrupt del touch.

Dopo l'inizializzazione, il touch può essere usato con i comandi:

- ILI TOUCH CALIBRATE
- ILI TOUCH SPRITE ...
- ILI TOUCH AREA ...

Se eseguito più volte, il comando chiude la precedente istanza del touch e la re-inizializza.
È necessario calibrare il touch al primo utilizzo tramite ILI TOUCH CALIBRATE.

Esempi pratici

Esempio 1 – Inizializzare il touch con CS=5 e rotazione 1

```
10 ILI INIT TOUCH 5 1
20 PRINT "Touch pronto!"
```

Output atteso (seriale):

Touch pronto!

Esempio 2 – Inizializzare e poi calibrare il touch

10 ILI INIT TOUCH 4 3
20 ILI TOUCH CALIBRATE

Comportamento atteso:

- Il touch viene inizializzato con CS=4 e rotazione 3.
- Parte la procedura interattiva di calibrazione (5 punti).
- I dati di calibrazione vengono salvati automaticamente.

Esempio 3 – Touch con SPI personalizzato (senza IRQ)

10 ILI INIT TOUCH 15 1 19 23 18

Esempio 4 – Touch con SPI e IRQ

10 ILI INIT TOUCH 15 1 19 23 18 27

Note

- **rotation** deve essere un numero da 0 a 3:
  - 0 → display verticale standard
  - 1 → display ruotato a destra
  - 2 → display capovolto
  - 3 → display ruotato a sinistra
- Se il touch non viene inizializzato, i comandi ILI TOUCH ... restituiranno errore.
- La calibrazione va fatta almeno una volta: senza, il mapping delle coordinate non sarà corretto.
- Se il tuo display **non ha il pin IRQ collegato**, **non è necessario specificarlo**.
- Puoi richiamare ILI INIT TOUCH più volte se cambi wiring o rotazione.

ILI LED

Sintassi:

ILI LED pin value

Descrizione:

Accende o spegne la **retroilluminazione** del display ILI9341 tramite il pin specificato.

Esempi pratici

```
10 ILI LED 32 1   ' Accende la retroilluminazione
20 ILI LED 32 0   ' Spegne la retroilluminazione
```

Note:

- Assicurarsi che il pin specificato sia connesso al LED backlight del display.
- value può essere 1 (accende) o 0 (spegne).
- Non tutti i moduli ILI9341 richiedono gestione manuale del LED.

ILI LINE

Sintassi:
ILI LINE <x1> <y1> <x2> <y2> [r g b] [drawnow]

Descrizione:
Disegna una linea tra due punti sullo schermo TFT.

- <x1> <y1>: punto iniziale.
- <x2> <y2>: punto finale.
- [r g b]: colore RGB opzionale (default bianco).
- [drawnow]: opzionale (1 = disegna subito, 0 = accoda).

Esempi pratici

Esempio 1 – Linea bianca immediata
10 ILI LINE 10 10 100 50
Disegna subito una linea bianca da (10,10) a (100,50).

Esempio 2 – Linea rossa differita
10 ILI LINE 20 60 120 60 255 0 0 0
20 ILI DRAW
La linea rossa viene effettivamente disegnata solo al comando ILI DRAW.

Note
• Le coordinate devono essere all'interno dei limiti del display (tipicamente 320x240).
• Il parametro [drawnow] è utile per disegnare più elementi contemporaneamente.

ILI PIXEL

Sintassi:
ILI PIXEL <x> <y> [r g b] [drawnow]

Descrizione:
Disegna un singolo pixel (punto) sullo schermo TFT alle coordinate specificate.

- <x> <y>: coordinate del pixel.
- [r g b]: colore RGB opzionale (valori 0–255). Se omessi, il colore è bianco.
- [drawnow]: opzionale.
  - 1 o assente → disegna subito.
  - 0 → accoda il pixel e verrà disegnato solo dopo ILI DRAW.

Esempi pratici

Esempio 1 – Pixel bianco immediato
10 ILI PIXEL 50 50 255 255 255
Disegna subito un pixel bianco alla posizione (50,50).

Esempio 2 – Pixel blu differito
10 ILI PIXEL 100 100 0 0 255 0
20 ILI DRAW
Il pixel blu in (100,100) viene disegnato solo alla riga 20, quando viene eseguito ILI DRAW.

Note
• Utile per test grafici o disegni a punti.
• Se le coordinate sono fuori dallo schermo, non viene disegnato nulla.
• Il parametro [drawnow] permette di costruire una schermata senza sfarfallii.

ILI RECT

Sintassi:
ILI RECT <x> <y> <w> <h> [r g b] [drawnow]

Descrizione:
Disegna un rettangolo vuoto (solo il bordo).

- <x> <y>: coordinate dell'angolo superiore sinistro.
- <w> <h>: larghezza e altezza del rettangolo.
- [r g b]: colore bordo (default bianco).
- [drawnow]: opzionale (1 = disegna subito, 0 = accoda).

Esempi pratici

Esempio 1 – Rettangolo bianco immediato
10 ILI RECT 20 20 100 50
Disegna subito un rettangolo vuoto 100x50 a (20,20).

Esempio 2 – Rettangolo rosso differito
10 ILI RECT 30 40 60 30 255 0 0 0
20 ILI DRAW
Il rettangolo rosso appare solo dopo l'esecuzione di ILI DRAW.

Note
• Solo i bordi vengono disegnati. Per riempirlo usa ILI FILLRECT.
• Il parametro [drawnow] consente di aggiornare più oggetti grafici insieme.

ILI SETBGCOLOR

Sintassi:

ILI SETBGCOLOR r g b

Descrizione:

Imposta il colore di sfondo predefinito usato da ILI SPRITE DRAW per riempire lo schermo prima di disegnare i singoli sprite.

- r g b: valori RGB da 0 a 255

Esempi pratici

Esempio 1 – Sfondo blu

```
10 ILI INIT 17 16 5 1
20 ILI SETBGCOLOR 0 0 255
30 ILI CLEAR
```

Risultato: lo schermo viene riempito di blu quando ILI CLEAR è eseguito.

Esempio 2 – Sfondo nero (default)

```
10 ILI SETBGCOLOR 0 0 0
```

Risultato: lo sfondo torna al nero.

Note

- Non cancella nulla da solo, ma viene applicato automaticamente alla prossima ILI CLEAR.
- Utile per creare animazioni con sfondi diversi o reset visivi tra frame.
- Se non usi SETBGCOLOR, il colore di sfondo è nero.

ILI SPEED

Sintassi

ILI SPEED mhz

Descrizione

Imposta la **frequenza del bus SPI** utilizzata per la comunicazione con il display TFT **ILI9341**.

La frequenza influisce direttamente sulla **velocità di aggiornamento grafico** (rettangoli, testo, sprite, fill, ecc.), ma **non modifica le prestazioni dell'interprete BASIC** o dei calcoli interni.

• **mhz**: frequenza SPI in **MHz**
Valori ammessi tipici: **40, 60, 80**

Il valore impostato viene memorizzato e utilizzato:

- durante le successive inizializzazioni del display (ILI INIT)
- immediatamente, se il display è già inizializzato

Esempi pratici

Esempio 1 – Impostazione a 40 MHz (valore consigliato)

```
10 ILI SPEED 40
20 ILI INIT 17 16 5 0
```

Risultato:
Il display viene inizializzato utilizzando una frequenza SPI di **40 MHz**, garantendo **massima stabilità** con la maggior parte dei cablaggi.

Esempio 2 – Aumento prestazioni grafiche (80 MHz)

```
10 ILI SPEED 80
20 ILI INIT 17 16 5 0
```

Risultato:
La comunicazione SPI viene portata a **80 MHz**, aumentando la velocità di operazioni grafiche come ILI FILLRECT, ILI TEXT e ILI CLEAR.
Richiede cablaggio corto e display compatibile.

Esempio 3 – Modifica della velocità a runtime

```
10 ILI INIT 17 16 5 0
20 ILI SPEED 80
30 ILI FILLRECT 0 0 320 240 0 255 0
```

Risultato:
La velocità SPI viene modificata **a caldo**, senza dover reinizializzare il display.

Note

• ILI SPEED **non inizializza il display**: deve essere usato **insieme a ILI INIT**.
• Se eseguito **prima** di ILI INIT, il valore viene applicato durante l'inizializzazione.
• Se eseguito **dopo** ILI INIT, la nuova velocità viene applicata immediatamente.
• Valori troppo elevati possono causare:

- artefatti grafici
- pixel corrotti
- instabilità dello schermo
  • In caso di problemi, ridurre la frequenza a **40 MHz**.

ILI SPRITE CLEAR

Sintassi:

ILI SPRITE CLEAR

Descrizione:

Rimuove tutti gli sprite attivi dalla memoria. Dopo questo comando, nessuno sprite verrà disegnato fino alla loro ricreazione con ILI SPRITE NEW.

Esempi pratici

Esempio 1 – Pulisce tutti gli sprite

10 ILI SPRITE CLEAR

Cancella visivamente lo schermo e libera la memoria sprite.

Note

- Utile per azzerare completamente una scena o inizializzare un nuovo stato grafico.

ILI SPRITE DELETE

Sintassi:

ILI SPRITE DELETE id

Descrizione:

Elimina lo **sprite** con identificatore id. Lo sprite non sarà più disegnato da ILI SPRITE DRAW.

Esempi pratici

10 ILI SPRITE DELETE 3

Rimuove lo sprite numero 3 dalla memoria.

Note:

- Dopo la cancellazione, lo slot dello sprite può essere riutilizzato con ILI SPRITE NEW.
- Non cancella il disegno dallo schermo finché non si esegue ILI SPRITE DRAW.

Sintassi:

ILI SPRITE DELETE id

Descrizione:

Rimuove lo sprite con identificativo id.

Esempi pratici

10 ILI SPRITE DELETE 2

Elimina lo sprite con ID 2.

ILI SPRITE DRAW

Sintassi:

ILI SPRITE DRAW
ILI SPRITE DRAW id

Descrizione:

Disegna tutti gli sprite attivi e visibili sulla schermata corrente. Deve essere chiamato dopo qualsiasi modifica agli sprite (creazione, spostamento, colore, contenuto, ecc.).

Esempi pratici

Disegna tutti gli sprite

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW
```

Visualizza un rettangolo rosso.

Disegna solo lo sprite richiesto

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW 0
```

Visualizza un rettangolo rosso.

Note

- Il comando ILI SPRITE DRAW aggiorna il contenuto del display in base allo stato corrente degli sprite.
- Il colore di sfondo viene impostato con ILI SETBGCOLOR (se specificato).
- Se non viene chiamato, gli sprite modificati non verranno ridisegnati.

ILI SPRITE FLIP

Sintassi

ILI SPRITE FLIP id modo

Descrizione

Il comando **ILI SPRITE FLIP** esegue uno specchiamento dell'immagine BITMAP.
Il parametro **modo** può essere:

- **H** → Flip orizzontale
- **V** → Flip verticale
- **B** → Flip completo (H + V)

Il carattere **non va tra virgolette**.
Se usato su sprite non BITMAP viene generato un errore.

Esempi pratici

Esempio 1 – Flip orizzontale

→ Specchia lo sprite orizzontalmente:

10 ILI SPRITE FLIP 0 H

Esempio 2 – Flip verticale

→ Specchia lo sprite verticalmente:

10 ILI SPRITE FLIP 0 V

Esempio 3 – Flip totale

→ Applica entrambi i flip:

10 ILI SPRITE FLIP 0 B

Esempio 4 – Modo non valido

→ Un carattere diverso da H, V o B genera errore:

10 ILI SPRITE FLIP 0 X
RUN

Output atteso:

?RUNTIME ERROR

ILI SPRITE FLIP: mode must be H,V or B.

Esempio 5 – Sprite non BITMAP

→ Se ID 3 è un Frame:

```
10 ILI SPRITE FLIP 3 H
RUN
```

Output atteso:

```
?RUNTIME ERROR
ILI SPRITE FLIP: sprite is not BITMAP.
```

Note

- Funziona solo su sprite BITMAP.
- Mantiene inalterati: posizione, dimensioni, scala, palette.
- Il flip modifica il contenuto interno del pattern.
- Ridisegna automaticamente lo sprite.

ILI SPRITE HIDE

Sintassi:

ILI SPRITE HIDE id

Descrizione:

Rende invisibile lo sprite specificato. Lo sprite non viene più disegnato finché non viene riattivato con ILI SPRITE SHOW.

Esempi pratici

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW
30 DELAY 2000
40 ILI SPRITE HIDE 0
50 ILI SPRITE DRAW
```

Il rettangolo rosso scompare dopo 2 secondi.

Note

- L’ID deve riferirsi a uno sprite esistente e attivo.
- Lo sprite non viene eliminato: resta in memoria ma non visibile.

ILI SPRITE MOVE

Sintassi:

ILI SPRITE MOVE id x y

Descrizione:

Sposta lo sprite esistente alle nuove coordinate (x, y).

Esempi pratici

```
10 ILI SPRITE NEW 2 RECT 10 100 30 30 0 255 255
20 ILI SPRITE DRAW
30 DELAY 2000
40 ILI SPRITE MOVE 2 80 100
50 ILI SPRITE DRAW
```

Un rettangolo azzurro si sposta a destra dopo 2 secondi.

Note

- Non modifica le dimensioni né il tipo dello sprite.
- Dopo lo spostamento, è necessario richiamare ILI SPRITE DRAW.

ILI SPRITE NEW

Sintassi generale

ILI SPRITE NEW id tipo x y parametri [r g b]

- **id** → identificatore univoco dello sprite
- **tipo** → RECT, FRAME, CIRCLE, LINE, CHAR, NUM, TEXT, BITMAP
- **x y** → posizione iniziale dello sprite
- **parametri** → variano in base al tipo
- **r g b** (opzionale) → colore in formato RGB (default = 255 255 255)

Gli sprite vengono disegnati tramite:

ILI SPRITE DRAW

---

1. RETTANGOLO PIENO (RECT)

Sintassi

ILI SPRITE NEW id RECT x y larghezza altezza [r g b]

Descrizione

Disegna un rettangolo riempito del colore indicato.

Esempio

10 ILI SPRITE NEW 0 RECT 20 20 60 40 255 0 0

→ Disegna un rettangolo **rosso**, 60×40, a **(20,20)**.

---

2. CORNICE (FRAME)

Sintassi

ILI SPRITE NEW id FRAME x y larghezza altezza [r g b]

Descrizione

Disegna solo il bordo di un rettangolo.

Esempio

20 ILI SPRITE NEW 1 FRAME 100 20 60 40 0 255 0

→ Disegna una cornice **verde**, 60×40, a **(100,20)**.

3. CERCHIO (CIRCLE)

Sintassi

ILI SPRITE NEW id CIRCLE x y raggio [r g b]

Descrizione

Disegna un cerchio pieno con centro in (x,y).

Esempio

30 ILI SPRITE NEW 2 CIRCLE 60 120 30 0 0 255

→ Disegna un cerchio **blu**, raggio 30, a **(60,120)**.

---

4. LINEA (LINE)

Sintassi

ILI SPRITE NEW id LINE x1 y1 x2 y2 [r g b]

Descrizione

Disegna una linea dal punto (x1,y1) a (x2,y2).

Esempio

40 ILI SPRITE NEW 3 LINE 20 180 120 200 255 255 0

→ Disegna una linea **gialla** da **(20,180)** a **(120,200)**.

---

5. CARATTERE SINGOLO (CHAR)

Sintassi

ILI SPRITE NEW id CHAR x y "carattere" dimensione [r g b]

Descrizione

Disegna un singolo carattere con font scalato.

Esempio

50 ILI SPRITE NEW 4 CHAR 160 40 "A" 3 0 255 255

→ Disegna **"A"**, grandezza 3, colore ciano, a **(160,40)**.

6. NUMERO (NUM)

Sintassi

ILI SPRITE NEW id NUM x y valore dimensione [r g b]

Descrizione

Disegna un numero con font scalato.
Supporta variabili.

Esempio

60 N = 123
70 ILI SPRITE NEW 5 NUM 160 80 N 2

→ Disegna **123** in dimensione 2, colore bianco.

---

7. TESTO (TEXT)

Sintassi

ILI SPRITE NEW id TEXT x y dimensione "testo" [r g b]

Descrizione

Disegna una stringa di testo.
Supporta concatenazioni e variabili stringa.

Esempio

80 A$ = "WORLD"
90 ILI SPRITE NEW 6 TEXT 20 240 2 "HELLO " + A$ 255 0 255

→ Disegna “HELLO WORLD”, grandezza 2, colore magenta.

---

8. BITMAP (BITMAP)

Sintassi

ILI SPRITE NEW id BITMAP x y larghezza altezza pattern_stringa scala

Descrizione

Disegna uno sprite grafico basato su una stringa di pattern, utilizzando una palette colori definita con:

ILI SPRITE PAL indice r g b

Esempio completo

Palette:

100 ILI SPRITE PAL 1 255 0 0
110 ILI SPRITE PAL 2 255 255 0

Pattern 5×5:

120 P$ = "01110" + "12221" + "12221" + "12221" + "01110"

Sprite:

130 ILI SPRITE NEW 7 BITMAP 40 40 5 5 P$ 4

→ Disegna un bitmap **5×5** scalato **×4**.

Disegnare gli sprite

Dopo aver creato uno o più sprite, per renderizzarli sul display:

ILI SPRITE DRAW

NOTE

- Il colore RGB è facoltativo (default: bianco).
- Gli sprite restano in memoria finché non vengono sovrascritti o eliminati.
- Il pattern dei bitmap deve contenere cifre corrispondenti agli indici della palette.
- Supportate variabili numeriche e stringa.

ILI SPRITE PAL

Sintassi:

ILI SPRITE PAL indice r g b

Descrizione:
Imposta un colore nella palette globale degli sprite BITMAP.
La palette contiene 16 colori (0–15).

0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F
Il colore **0** è sempre trasparente.

Esempi pratici

Definizione di una palette base

```
10 ILI SPRITE PAL 0 0 0 0      ' Trasparente
20 ILI SPRITE PAL 1 255 0 0    ' Rosso
30 ILI SPRITE PAL 2 0 255 0    ' Verde
40 ILI SPRITE PAL 3 0 0 255    ' Blu
50 ILI SPRITE PAL 4 255 255 0  ' Giallo
```

Note

- Usata da tutti gli sprite **BITMAP** del programma.
- Ogni sprite bitmap interpreta i caratteri 1..F come indici di questa palette.
- Il colore 0 non viene disegnato (trasparenza).

ILI SPRITE ROTATE

Sintassi

ILI SPRITE ROTATE id gradi

Descrizione

Il comando **ILI SPRITE ROTATE** ruota uno sprite di tipo **BITMAP** di un numero di gradi specificato.

Il comando modifica internamente la mappa dei pixel e ridisegna automaticamente lo sprite.

Se lo sprite non è di tipo BITMAP, viene generato un errore.

Esempi pratici

Esempio 1 – Ruotare uno sprite di 90°

→ Ruota lo sprite BITMAP con ID 0 di 90 gradi in senso orario:

10 ILI SPRITE ROTATE 0 90

Esempio 2 – Ruotarlo di 180°

→ Ruota lo sprite 5 capovolgendolo completamente:

10 ILI SPRITE ROTATE 5 180

Esempio 3 – Usato su sprite non BITMAP

→ Se ID 2 è un RECT, il comando fallisce:

10 ILI SPRITE ROTATE 2 90
RUN

Output atteso:

?RUNTIME ERROR
ILI SPRITE ROTATE: sprite is not BITMAP

Note

- Funziona **solo** su sprite creati con ILI SPRITE NEW ... BITMAP.
- Mantiene posizione, dimensioni e scala invariati.
- Aggiorna automaticamente il disegno sul display.

ILI SPRITE SETBITMAP

Sintassi:

ILI SPRITE SETBITMAP id pattern$

Descrizione:
Cambia il pattern grafico di uno sprite BITMAP già creato.
Serve per animazioni (camminata, movimento, sprite del serpente, oggetti che cambiano forma…).

Esempi pratici

Animazione semplice (2 fotogrammi)

```
10 P1$ = "01110" + "10001" + "11111" + "10001" + "01110"
20 P2$ = "01110" + "11011" + "11111" + "11011" + "01110"

30 ILI SPRITE PAL 1 255 255 255
40 ILI SPRITE NEW 0 BITMAP 40 40 5 5 P1$ 4

50 FOR I = 1 TO 10
60   ILI SPRITE SETBITMAP 0 P1$
70   ILI SPRITE DRAW 0
80   DELAY 100
90   ILI SPRITE SETBITMAP 0 P2$
100  ILI SPRITE DRAW 0
110  DELAY 100
120 NEXT I
```

Note

- Lo sprite mantiene: posizione, scala, dimensioni, visibilità.
- Cambia *solo* il pattern.
- Il nuovo pattern deve avere la stessa dimensione W×H dello sprite originario.
- Per vedere il cambio serve un ILI SPRITE DRAW id.

ILI SPRITE SETCHAR

Sintassi:

ILI SPRITE SETCHAR id "carattere" r g b

Descrizione:

Modifica il carattere visualizzato dallo sprite di tipo CHAR.

Esempi pratici

```
10 ILI SPRITE SETCHAR 4 "B" 255 255 0
20 ILI SPRITE DRAW
```

Cambia il carattere nello sprite 4 in “B” giallo.

Note

- Funziona solo su sprite di tipo CHAR.

ILI SPRITE SETNUM

Sintassi:

ILI SPRITE SETNUM id valore r g b

Descrizione:

Modifica il numero visualizzato da uno sprite di tipo NUM.

Esempi pratici

```
10 ILI SPRITE SETNUM 5 2024 255 105 180
20 ILI SPRITE DRAW
```

Visualizza il numero 2024 in rosa.

Note

- Funziona solo su sprite NUM.
- Il numero è trattato come testo.

ILI SPRITE SETTEXT

Sintassi:

ILI SPRITE SETTEXT <id> <testo/expr> [r g b]

Descrizione:
Cambia il contenuto e (opzionalmente) il colore dello sprite di tipo **TEXT**.
Il parametro <testo/expr> ora può essere:

- una stringa tra virgolette ("HELLO")
- una variabile stringa (A$)
- un'espressione concatenata ("VAL=" + N + " " + A$)
- una funzione stringa (STR$(), LEFT$(), ecc.)
- un numero o un'espressione numerica (convertita automaticamente in stringa)

I valori [r g b] sono opzionali (default bianco).

Esempi pratici

Esempio 1 – Testo semplice e colore

```
10 ILI SPRITE SETTEXT 6 "Cambiato testo" 0 255 0
20 ILI SPRITE DRAW
```

Cambia il testo nello sprite 6 in Cambiato testo verde.

Esempio 2 – Concatenazione con variabile numerica

```
10 N = 42
20 ILI SPRITE SETTEXT 1 "VAL=" + N 255 0 0
30 ILI SPRITE DRAW 1
```

Cambia lo sprite 1 mostrando VAL=42 in rosso.

Esempio 3 – Variabile stringa

```
10 A$ = "WORLD"
20 ILI SPRITE SETTEXT 2 "HELLO " + A$
30 ILI SPRITE DRAW 2
```

Cambia lo sprite 2 mostrando HELLO WORLD.

Note

- Funziona solo su sprite di tipo **TEXT**.
- Non è più obbligatorio usare virgolette doppie: puoi concatenare stringhe, variabili e numeri con +.
- Colore [r g b] opzionale (default bianco).
- Dopo la modifica, usa ILI SPRITE DRAW o ILI SPRITE DRAW id per ridisegnare lo sprite.

ILI SPRITE SHOW

ILI SPRITE SHOW id

Descrizione:

Rende di nuovo visibile uno sprite precedentemente nascosto con ILI SPRITE HIDE.

Esempi pratici

```
10 ILI SPRITE SHOW 1
20 ILI SPRITE DRAW
```

Lo sprite 1 torna visibile.

Note

- La posizione, contenuto e colore non vengono modificati.

ILI SPRITE CLEAR

Sintassi:

ILI SPRITE CLEAR

Descrizione:

Rimuove tutti gli sprite attivi. Lo schermo resta vuoto fino alla creazione di nuovi sprite.

Esempi pratici

10 ILI SPRITE CLEAR

Tutti gli sprite vengono eliminati.

ILI SPRITE COLLISION

Sintassi

ILI SPRITE COLLISION id1 TO id2 GOTO line
ILI SPRITE COLLISION id1 TO ANY GOTO line
ILI SPRITE COLLISION id1 TO id2 SET variabile
ILI SPRITE COLLISION id1 TO ANY SET variabile

Descrizione

Verifica la collisione **rettangolare** (AABB) fra sprite creati con ILI SPRITE NEW.
In caso di collisione:

- con **GOTO** salta alla riga indicata;
- con **SET variabile** assegna 1 (collisione) o 0 (no collisione) a una variabile numerica o stringa ($).

Esempio – GOTO (rettangolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
45 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE"
210 END
```

Esempio – SET variabile (rettangolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
45 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
```

```
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO 2 SET C
67 IF C THEN PRINT "Colpito!" ELSE PRINT "Libero"
68  IF C THEN ILI SPRITE DRAW 2
70 GOTO 53
```

Esempio – ANY con SET (rettangolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 RECT 10 100 10 10 255 0 0
41 ILI SPRITE NEW 2 RECT 120 100 10 10 0 255 0
42 ILI SPRITE NEW 3 RECT 200  40 10 10 0 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISION 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 ha urtato qualcuno!"
68 IF HIT THEN ILI SPRITE DRAW 2
70 GOTO 53
```

Note

- Gli **ID** devono riferirsi a sprite **attivi** (ILI SPRITE NEW).
- Variabili: se terminano con $ → "1"/"0"; altrimenti numeriche → 1.0/0.0.
- AABB è rapido e adatto alla maggior parte dei giochi.
- Con molti sprite, limita i test per ciclo per mantenere il frame rate.

ILI SPRITE COLLISIONC

Sintassi

ILI SPRITE COLLISIONC id1 TO id2 GOTO line
ILI SPRITE COLLISIONC id1 TO ANY GOTO line
ILI SPRITE COLLISIONC id1 TO id2 SET variabile
ILI SPRITE COLLISIONC id1 TO ANY SET variabile

Descrizione

Verifica la collisione **circolare** usando i cerchi **inscritti** nei bounding box degli sprite:

- centro = centro del bbox; raggio = min(larghezza,altezza)/2.
  Utile per sprite rotondi (palle, monete) o per una rilevazione più “morbida”.

Esempi pratici (copiabili)

Esempio – GOTO (circolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 10 255 0
45 ILI SPRITE NEW 2 CIRCLE 120 100 10 10 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE CIRCOLARE"
210 END
```

Esempio – SET variabile (circolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 10 255 0
45 ILI SPRITE NEW 2 CIRCLE 120 100 10 10 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
```

```
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO 2 SET CIRC
67 IF CIRC THEN PRINT "Contatto (cerchi)!" ELSE PRINT "Nessun contatto"
68 IF CIRC THEN ILI SPRITE DRAW 2
70 GOTO 53
```

Esempio – ANY con SET (circolare)

```
10 ILI INIT 17 16 5 3
15 ILI LED 32 1
20 ILI SETBGCOLOR 0 0 0
30 ILI CLEAR
40 ILI SPRITE NEW 1 CIRCLE 10 100 10 0 255 0
41 ILI SPRITE NEW 2 CIRCLE 120 100 10 255 0 0
42 ILI SPRITE NEW 3 CIRCLE 200  40 10 0 0 255
50 ILI SPRITE DRAW
51 X = 10
52 Y = 100
53 X = X+1
60 ILI SPRITE MOVE 1 X Y
61 DELAY 30
65 ILI SPRITE DRAW 1
66 ILI SPRITE COLLISIONC 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 ha toccato almeno uno (circolare)"
68 IF HIT THEN ILI SPRITE DRAW 2
70 GOTO 53
```

Note

- Richiede sprite **attivi** e con bounding box valido.
- Variabili: numeriche (1/0) o stringa ("1"/"0").
- COLLISIONC è leggermente più costoso di COLLISION, ma utile con sprite tondeggianti.
- Mantieni cadenza costante: se fai molti test, distribuiscili su più frame.

ILI TEXT

Sintassi:
ILI TEXT <x> <y> <size> <testo/expr> [r g b] [drawnow]

Descrizione:
Visualizza un testo sul display alla posizione specificata, con dimensione e colore.
Il parametro <testo/expr> può essere:
• una stringa tra virgolette ("HELLO")
• una variabile stringa (A$)
• un'espressione concatenata con + (es. "CIAO " + A$ + N)
• una funzione stringa (LEFT$(A$,3), STR$(X))
• un numero o un'espressione numerica (convertita in stringa)

- [r g b]: colore del testo (default bianco).
- [drawnow]: opzionale (1 = stampa subito, 0 = accoda per ILI DRAW).

Esempi pratici

Esempio 1 – Testo fisso immediato
10 ILI TEXT 10 50 2 "HELLO ILI" 0 255 255
Scrive "HELLO ILI" in ciano, grandezza 2, alla posizione (10,50).

Esempio 2 – Testo concatenato con variabile stringa
10 A$ = "WORLD"
20 ILI TEXT 20 80 2 "HELLO " + A$
Mostra "HELLO WORLD" in bianco.

Esempio 3 – Testo rosso differito
10 N = 123
20 ILI TEXT 10 120 2 "VALUE=" + N 255 0 0 0
30 ILI DRAW
Scrive "VALUE=123" in rosso solo al comando ILI DRAW.

Note
• È possibile concatenare testo, numeri e variabili.
• [r g b] è opzionale (0–255).
• [drawnow]=0 accoda il testo per disegnarlo insieme ad altri elementi.
• Il testo resta visibile finché non si esegue ILI CLEAR.

ILI TOUCH AREA

Sintassi

ILI TOUCH AREA x y w h GOTO line [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH AREA x y w h SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]

Descrizione

Verifica se il tocco cade dentro l'area rettangolare specificata (coordinate schermo ILI, dopo calibrazione).

Con le opzioni puoi distinguere **quando** scatta l'evento:

Opzione	Significato
(nessuna)	compatibilità: scatta finché l'area è toccata
DOWN	solo al primo tocco (touch down)
UP	solo al rilascio (touch up)
HOLD ms	dopo che il dito resta nell'area per almeno *ms* millisecondi
DEBOUNCE ms	impone un tempo minimo (ms) tra due attivazioni consecutive

L'area accetta dimensioni negative (w, h): viene normalizzata automaticamente.

Esempi pratici

Area con salto

```
10 X = 100
20 Y = 180
30 W = 120
40 H = 40
50 ILI FILLRECT X Y W H 0 120 200
60 ILI TOUCH AREA X Y W H GOTO 200 DOWN DEBOUNCE 200
70 GOTO 60
200 PRINT "Tap nell'area!"
210 END
```

Area con variabile

```
10 ILI TOUCH AREA 20 20 100 60 SET T DOWN
20 IF T THEN PRINT "Tap nella box 20,20,100,60"
30 GOTO 10
```

Note

- Se il touch non è disponibile o non è premuto, **non avviene alcun salto** e con SET la variabile riceve 0.
- Le coordinate del tocco sono mappate in base a **rotazione** e **calibrazione** (minRawX/maxRawX/minRawY/maxRawY).
- Usa DEBOUNCE per evitare ripetizioni indesiderate se tieni premuto il dito.
- Usa HOLD per creare pulsanti “a pressione lunga”.

ILI TOUCH CALIBRATE

Sintassi:

ILI TOUCH CALIBRATE
ILI TOUCH CALIBRATE FORCE

Descrizione:
Avvia una procedura guidata di calibrazione del touch. Il sistema ti chiede di toccare alcuni punti agli angoli dello schermo per misurare i valori grezzi del pannello (min/max su X e Y) e allineare le coordinate del tocco alle coordinate TFT.

- Esegue la calibrazione **nella rotazione corrente** del display.
- Aggiorna i parametri interni usati per il mapping (tipicamente minRawX, maxRawX, minRawY, maxRawY).
- Richiede che il touch sia già inizializzato con ILI INIT TOUCH ....

Esempi pratici

Esempio 1 – Calibrazione semplice

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 10 20 2 "Calibrazione completata" 0 255 0
```

Risultato: parte la procedura; al termine i tocchi risultano allineati alle coordinate schermo.

Esempio 2 – Calibrazione + test puntatore

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 15 0 2 "Tocca il quadrato" 255 255 255
70 ILI SPRITE NEW 0 RECT 90 40 50 50 255 0 0
80 ILI SPRITE DRAW
90 ILI TOUCH SPRITE 0 GOTO 200
100 GOTO 90
200 ILI TEXT 25 120 2 "Tocco rilevato!" 0 255 0
210 END
```

Esempio 3 – Calibrazione + test doppio sprite

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 IF TOUCH CALIBRATE = 0 THEN ILI TOUCH CALIBRATE
50 ILI CLEAR
60 ILI TEXT 15 0 2 "Tocca un quadrato" 255 255 255
70 ILI SPRITE NEW 0 RECT 90 40 50 50 255 0 0
80 ILI SPRITE NEW 1 RECT 90 100 50 50 0 0 255
90 ILI SPRITE DRAW
100 ILI TOUCH SPRITE 0 GOTO 200
110 ILI TOUCH SPRITE 1 GOTO 220
120 GOTO 100
200 ILI TEXT 30 180 2 "Toccato rosso!" 0 255 0
210 END
220 ILI TEXT 30 180 2 "Toccato blu!" 0 255 0
230 END
```

Esempio 4 – Forza la calibrazione anche se già presente

```
10 ILI INIT 17 16 5 2
20 ILI LED 32 1
30 ILI INIT TOUCH 13 2
40 ILI TOUCH CALIBRATE FORCE
```

Note:

- Esegui ILI TOUCH CALIBRATE **dopo** ILI INIT e ILI INIT TOUCH.
- Non appoggiare il dito sullo schermo **prima** dell'avvio: tocca solo quando richiesto.
- Se cambi la **rotazione** del display, ripeti la calibrazione.

ILI TOUCH SPRITE

Sintassi

ILI TOUCH SPRITE id GOTO line [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH SPRITE id SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]
ILI TOUCH SPRITE ANY SET var [DOWN|UP|HOLD ms|DEBOUNCE ms]

Descrizione

Rileva il tocco sul **bounding box** di uno sprite.

Variante	Azione
id GOTO line	salta a line quando lo sprite id viene toccato
id SET var	assegna 1 o 0 a var se il tocco cade dentro lo sprite
ANY SET var	assegna a var l'ID del primo sprite toccato, o 0 se nessuno

Con le opzioni puoi distinguere **quando** scatta l'evento:

Opzione	Significato
(nessuna)	compatibilità: scatta finché l'area è toccata
DOWN	solo al primo tocco (touch down)
UP	solo al rilascio (touch up)
HOLD ms	dopo che il dito resta nell'area per almeno *ms* millisecondi
DEBOUNCE ms	impone un tempo minimo (ms) tra due attivazioni consecutive

Funziona con tutti i tipi di sprite (RECT, FRAME, CIRCLE, LINE, TEXT, CHAR, NUM, PIXELS…).

Esempi pratici

Singolo sprite con variabile 1/0

```
10 ILI SPRITE NEW 5 RECT 40 40 50 30 0 200 0
20 ILI SPRITE DRAW
30 ILI TOUCH SPRITE 5 SET HIT DOWN DEBOUNCE 150
40 IF HIT THEN PRINT "Hai toccato lo sprite 5"
50 GOTO 30
```

Qualunque sprite → variabile = ID

```
10 ILI TOUCH SPRITE ANY SET S DOWN
20 IF S > 0 THEN PRINT "Sprite toccato: "; S
30 GOTO 10
```

Sprite con salto

```
10 ILI SPRITE NEW 3 RECT 100 100 80 40 255 0 0
20 ILI SPRITE DRAW
30 ILI TOUCH SPRITE 3 GOTO 200 DOWN
40 GOTO 30
200 PRINT "Toccato lo sprite 3!"
210 END
```

Note

- Gli sprite devono essere **attivi e visibili**, altrimenti il tocco è ignorato.
- DOWN, UP, HOLD, DEBOUNCE funzionano anche in combinazione (es. DOWN DEBOUNCE 200).
- Il test usa il **bounding box** (x,y,w,h).
  Per test realmente circolari puoi usare ancora COLLISIONC se necessario.
- Se il touch non è disponibile o non è premuto, non avviene alcun salto e le variabili SET valgono 0.

INCLUDE

Sintassi

INCLUDE "nomefile.bas"
INCLUDE SD "nomefile.bas"
INCLUDE SPIFFS "nomefile.bas"

Descrizione

Il comando **INCLUDE** inserisce nel programma BASIC il contenuto di un file .bas presente su SPIFFS o SD.

Il file incluso viene:

- letto e compilato riga per riga,
- rinumerato automaticamente senza conflitti,
- aggiunto al programma in memoria,
- reso disponibile per l'esecuzione (funzioni, codice, ecc.).

Il contenuto incluso **non viene mostrato nel comando LIST**, che visualizza solo la riga INCLUDE.

Quando la riga INCLUDE viene cancellata, **tutto il codice caricato da quel file viene rimosso automaticamente**.

Questo comando è utile per:

- suddividere il programma in moduli
- creare livelli di gioco separati
- includere sprite, routine o funzioni esterne
- mantenere il codice principale più pulito

Esempi

Includere un modulo da SPIFFS

```
10 PRINT "INIZIO"
20 INCLUDE "modulo.bas"
30 CALLFUNC TEST
40 END
```

Contenuto del file modulo.bas

```
10 FUNC TEST
20   PRINT "funziona!"
30 ENDFUNC
```

Output:

INIZIO
funziona!

Includere da SD

20 INCLUDE SD "livello2.bas"

Eliminare l’INCLUDE e rimuovere il codice associato

20

Questo rimuove:

- la riga 20 INCLUDE ...
- tutte le righe generate dall’INCLUDE
- le funzioni definite nel file incluso

Note

- Il listato mostra solo INCLUDE, non il contenuto del file.
- Le funzioni definite nei file inclusi possono essere richiamate normalmente.
- Gli include possono essere annidati fino a 8 livelli.
- Cambiare il numero di riga dell’INCLUDE aggiorna correttamente tutte le righe caricate.
- Usare NEW per cancellare completamente il programma.

INITRTC

Sintassi:

INITRTC <indirizzo_hex>

 Prima di usare INITRTC è obbligatorio chiamare il comando WIRE
per specificare i pin SDA e SCL del bus I2C.

Descrizione:

Il comando **INITRTC** inizializza il modulo RTC (Real Time Clock) collegato al bus I2C.

- Accetta come parametro **l'indirizzo I2C del modulo RTC**, in forma decimale o esadecimale (0x68).
- Controlla che il dispositivo risponda a quell'indirizzo.
- Avvia la libreria RTC e, se il modulo è configurato correttamente, sincronizza l'orario del sistema con quello dell'RTC.

Viene utilizzato per gestire:

- LETTURA ORA (TIMEH, TIMEM, TIMES)
- LETTURA DATA (DATEY, DATEM, DATED)

e altri comandi che dipendono dal tempo reale.

Esempi pratici

Esempio 1 – Inizializzare l'RTC e stampare l'ora

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT TIMEH; ":"; TIMEM; ":"; TIMES
RUN
```

Output atteso:

14:03:52

Esempio 2 – Verificare la presenza dell'RTC

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT "RTC inizializzato correttamente!"
```

RUN

Nota:

- **WIRE è obbligatorio** prima di INITRTC → se non è stato chiamato, viene generato un errore.
- L'indirizzo tipico dell'RTC DS1307/DS3231 è **0x68**.
- Il modulo deve essere collegato correttamente ai pin SDA e SCL indicati da WIRE.
- L'RTC deve essere alimentato.
- Funziona con moduli DS1307, DS3231 e compatibili.

INITSD

Sintassi

INITSD <cs> <mosi> <miso> <sck> [0|1] [<cs_alt1> <cs_alt2> ...]

Descrizione

Il comando **INITSD** inizializza la scheda SD specificando i pin da utilizzare per l'interfaccia SPI.

Argomento	Descrizione
cs	Pin Chip Select della scheda SD
mosi	Pin MOSI (Master Out Slave In)
miso	Pin MISO (Master In Slave Out)
sck	Pin SCK (Clock SPI)
[0 | 1] *(opzionale)*	Se indicato, salva le preferenze su SPIFFS in /prefs.cfg: 0 = non avviare automaticamente la SD all'avvio 1 = avvia automaticamente la SD all'avvio
[<cs_alt1> <cs_alt2> ...] *(opzionale)*	Elenco di pin **Chip Select** di altri dispositivi SPI (display, touch, sensori, ecc.) da portare **HIGH** durante l'inizializzazione e memorizzare nelle preferenze

Quando viene specificato l'argomento [0|1], il comando salva in /prefs.cfg:

- i pin usati per la SD (CS, MOSI, MISO, SCK);
- la scelta di auto-mount (SD_AUTO);
- gli eventuali pin CS aggiuntivi (EXTRA_CS).

All'avvio, se SD_AUTO=1, il firmware:

1. imposta **HIGH** tutti i pin elencati in EXTRA_CS;
2. monta automaticamente la SD con i pin salvati;
3. mostra lo spazio libero disponibile.

Esempi pratici

Esempio 1 – Inizializzare una SD con pin personalizzati

10 INITSD 13 23 19 18

```
20 PRINT SDFREE
RUN
```

Output atteso:
Mostra lo spazio libero sulla SD se inizializzata correttamente.

Esempio 2 – Inizializzare una SD e salvare le preferenze per l'avvio automatico

```
10 INITSD 13 23 19 18 1
20 PRINT SDFREE
RUN
```

Output atteso:
La SD viene inizializzata, le impostazioni vengono salvate in /prefs.cfg,
e la scheda sarà montata automaticamente ai successivi avvii.

Esempio 3 – Inizializzare una SD con altri dispositivi SPI collegati

```
10 INITSD 13 23 19 18 1 17 4
20 PRINT SDFREE
RUN
```

Spiegazione:

- 13 23 19 18 → pin SD (CS, MOSI, MISO, SCK)
- 1 → abilita auto-mount e salva le preferenze
- 17 4 → porta HIGH i pin CS del display ILI (17) e del touch (4)

Output atteso:
Mostra lo spazio libero sulla SD e salva le preferenze in /prefs.cfg.

Note

- È necessario eseguire INITSD prima di utilizzare comandi che accedono ai file SD (es. LOAD, DIR SD, SDFREE), a meno che non sia attivo l'auto-mount.
- La scheda SD deve essere **formattata FAT32**.
- La corretta assegnazione dei pin dipende dal cablaggio hardware.
- Dopo l'inizializzazione la SD è pronta per letture e scritture.
- Per verificare lo stato della SD e lo spazio disponibile: PRINT SDFREE
- Il file di configurazione /prefs.cfg viene usato all'avvio per gestire l'inizializzazione automatica.

INPUT

Sintassi:

INPUT variabile

Descrizione:
Il comando INPUT consente di **richiedere un valore da tastiera** (tramite terminale seriale).
Può essere usato per ricevere sia:

- **valori numerici** (es: A, X)
- **stringhe** (es: NOME$, TITOLO$)

L'esecuzione si **ferma finché l'utente non inserisce un valore** e preme INVIO.
Non è necessario specificare il tipo: il BASIC distingue automaticamente in base alla variabile ($ = stringa).

Esempi pratici

Esempio 1 – Richiesta di un numero intero

```
10 INPUT A
20 PRINT "HAI INSERITO: "; A
RUN
```

Output atteso (se inserisci 42):

```
? 42
HAI INSERITO: 42
```

Esempio 2 – Richiesta di una stringa

```
10 INPUT NOME$
20 PRINT "CIAO "; NOME$
RUN
```

Output atteso (se inserisci MARCO):

```
? MARCO
CIAO MARCO
```

Esempio 3 – INPUT multiplo (con due righe)
→ È necessario usare più istruzioni per più valori:

```
10 INPUT A
20 INPUT B
30 PRINT "SOMMA: "; A + B
RUN
```

Output atteso (es. input 5 e 7):

```
? 5
? 7
SOMMA: 12
```

Esempio 4 – Validazione semplice
→ Verifica se il valore inserito è positivo:

```
10 INPUT X
20 IF X < 0 THEN PRINT "VALORE NEGATIVO" ELSE PRINT "OK"
RUN
```

Esempio 5 – Con messaggio personalizzato
→ Aggiungi un prompt visivo con PRINT prima:

```
10 PRINT "INSERISCI IL TUO NOME:"
20 INPUT N$
30 PRINT "BENVENUTO "; N$
RUN
```

Output atteso:

```
INSERISCI IL TUO NOME:
? LUCA
BENVENUTO LUCA
```

Nota:

- Il simbolo ? è mostrato automaticamente come prompt di input

INT(x)

Sintassi:

INT(x)

Descrizione:
La funzione INT(x) restituisce la **parte intera** di un numero x, **tronca i decimali** e **arrotonda verso lo zero**.
È utile per convertire un numero decimale in intero in modo controllato, ad esempio per gestire cicli, indici di array, arrotondamenti personalizzati.

- INT(4.7) ⟶ 4
- INT(-2.3) ⟶ -2

 Non confondere con un arrotondamento: INT **non** arrotonda per eccesso o difetto — taglia semplicemente i decimali.

Esempi pratici

Esempio 1 – Parte intera di un numero positivo

```
10 A = 5.89
20 PRINT "INT(5.89) = "; INT(A)
RUN
```

Output atteso:

```
INT(5.89) = 5
```

Esempio 2 – Parte intera di un numero negativo

```
10 PRINT "INT(-3.99) = "; INT(-3.99)
RUN
```

Output atteso:

```
INT(-3.99) = -3
```

Esempio 3 – Uso per accedere a indici di array
⟶ Elimina il decimale da una divisione e usa il risultato come indice:

```
10 DIM A(10)
20 X = 5.7
30 A(INT(X)) = 99
40 PRINT A(5)
RUN
```

Output atteso:

99

Esempio 4 – Uso in un ciclo per numeri casuali interi
→ Genera 10 numeri casuali da 0 a 9:

```
10 FOR I = 1 TO 10
20 PRINT INT(RND(10))
30 NEXT I
RUN
```

Output atteso:
(esempio)

```
7
2
9
0
5
...
```

IP

Sintassi:
IP

Descrizione:
Il comando IP stampa l'indirizzo IP corrente del dispositivo se è connesso a una rete Wi-Fi tramite il comando WIFI.
Serve per visualizzare facilmente l'indirizzo con cui il dispositivo è raggiungibile nella rete locale.

Esempi pratici

Esempio 1 – Connessione Wi-Fi e stampa IP

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 PRINT IP
RUN
```

Output atteso:

```
192.168.1.42
```

Esempio 2 – Attendere e poi visualizzare IP

```
5 WIFI "ssid" "password"
10 DELAY 5000
20 PRINT "IL MIO IP È:"
30 IP
RUN
```

Output atteso:

```
IL MIO IP È:
192.168.1.50
```

Nota:
• Mostra l'indirizzo IP ottenuto tramite DHCP
• Funziona solo dopo aver eseguito con successo WIFI

IPAP

Sintassi:
IPAP

Descrizione:
Il comando IPAP stampa l'indirizzo IP locale del dispositivo quando è in modalità Access Point (creata tramite il comando AP).
Questo è utile per sapere dove accedere al dispositivo quando ha creato una rete Wi-Fi propria.

Esempi pratici

Esempio 1 – Avviare un Access Point e vedere l'IP

```
10 WIFIAP "Basic32AP" "mypassword"
20 DELAY 2000
30 PRINT IPAP
RUN
```

Output atteso:

```
192.168.4.1
```

Nota:
• Mostra l'indirizzo IP del dispositivo come hotspot
• L'IP predefinito è solitamente 192.168.4.1
• Funziona solo dopo WIFIAP

KEYPOLL

Sintassi

KEYPOLL ON
KEYPOLL OFF

Descrizione

KEYPOLL abilita o disabilita il polling della tastiera durante l'esecuzione del programma.

Agisce contemporaneamente su:

- tastiera PS/2
- tastiera I²C CardKB

Quando il polling è disabilitato:

- il programma non rileva tasti
- ESC non ferma il RUN
- eventuali funzioni come CARDKB non restituiscono caratteri

È utile per **massimizzare la velocità** di giochi o calcoli intensivi.

Comportamento

KEYPOLL ON

Riattiva il polling tastiere.
Usato per rientrare in un menu o accettare input.

KEYPOLL OFF

Disabilita la lettura tastiere.
Migliora le prestazioni riducendo il carico CPU.

Interazione con RUN FAST

- RUN FAST parte automaticamente con KEYPOLL OFF.
- Il programma può riattivare input con KEYPOLL ON.

Esempi pratici

Esempio 1 — Gioco veloce senza tastiera

```
10 KEYPOLL OFF        ' massima velocità
20 FOR I = 1 TO 20000
30   ' logica di gioco
40 NEXT I
50 END
```

Esempio 2 — Menu con input

```
10 KEYPOLL OFF          ' gioco veloce
20 GOSUB 500
30 KEYPOLL ON           ' riattivo tastiera
40 GOSUB 900            ' menu
50 END
```

Esempio 3 — RUN FAST + abilitazione selettiva

RUN FAST

Programma:

```
10 REM Ciclo massimo veloce
20 FOR I = 1 TO 30000: NEXT

30 REM Ora richiedo input → riattivo polling
40 KEYPOLL ON
50 PRINT "Premi un tasto..."
60 K = CARDKB
70 IF K = 0 THEN GOTO 60
80 PRINT "Hai premuto:", CHR$(K)
```

Note

- KEYPOLL non influenza BREAKPIN: rimane funzionante.
- Disabilitare il polling può aumentare la velocità del RUN fino al 20–30%.
- PS/2 e CardKB sono gestite entrambe da questo comando.
- KEYPOLL OFF non influisce sulle funzioni che leggono file o sensori.

KPAVAILABLE

Sintassi:
KPAVAILABLE

Descrizione:
Ritorna il **numero di tasti** attualmente nel buffer (0 se vuoto). Può essere usato in PRINT, IF, LET.

Esempi pratici

Esempio 1 – IF

```
10 IF KPAVAILABLE > 0 THEN KPREAD K$
RUN
```

Esempio 2 – PRINT

```
10 PRINT KPAVAILABLE
RUN
```

Esempio 3 – LET

```
10 LET N = KPAVAILABLE
20 PRINT "BUFFER=";N
RUN
```

Nota:
• Utile per **polling** non bloccante.
• Valore intero ≥ 0.

KPFLUSH

Sintassi:
KPFLUSH

Descrizione:
Svuota il **buffer** dei tasti (scarta gli eventi pendenti). Utile prima di leggere un input “pulito”.

Esempi pratici

Esempio 1 – Pulizia prima di PIN

```
10 KPFLUSH
20 PRINT "Inserisci PIN e premi #"
30 KPWAIT K$
40 IF K$="#" THEN PRINT "FINE" : END
50 PRINT K$;
60 GOTO 30
RUN
```

Esempio 2 – Reset buffer tra schermate

```
10 PRINT "Schermata 1"
20 KPWAIT _
30 KPFLUSH
40 PRINT "Schermata 2"
RUN
```

Nota:
• Non richiede parametri.
• Non modifica la configurazione del keypad.

KPINIT (Matrix keypad)

Sintassi:
KPINIT nrows ncols r1 r2 ... rn c1 c2 ... cm

Descrizione:
Inizializza il tastierino matriciale specificando **numero di righe** e **colonne**, poi l'elenco dei **pin delle righe** (OUTPUT) e dei **pin delle colonne** (INPUT_PULLUP), nell'ordine.

Esempi pratici

Esempio 1 – 2×2

```
10 KPINIT 2 2 13 12 14 27
20 PRINT "KEYPAD PRONTO"
RUN
```

Output atteso:

```
KEYPAD PRONTO
```

Esempio 2 – 3×4 classico

```
10 KPINIT 4 3 2 3 4 5   6 7 8
RUN
```

Esempio 3 – 4×4

```
10 KPINIT 4 4 2 3 4 5   6 7 8 9
RUN
```

Nota:
• Va chiamato **prima** di KPMAP/KPREAD/KPWAIT.
• Le **righe** sono OUTPUT (scansione), le **colonne** INPUT_PULLUP.
• Il numero totale di pin indicati deve essere nrows + ncols.

KPMAP

Sintassi:
KPMAP "stringaMappa"

Descrizione:
Imposta la **mappa dei tasti** in ordine **riga-per-riga**. La lunghezza della stringa deve essere nrows * ncols. Se non impostata, verranno restituiti codici posizione tipo "R1C2".

Esempi pratici

Esempio 1 – 2×2 → "ABCD"

```
10 KPMAP "ABCD"
RUN
```

Esempio 2 – 4×4 standard

```
10 KPMAP "123A456B789C*0#D"
RUN
```

Esempio 3 – 3×4 numerico

```
10 KPMAP "123456789*0#"
RUN
```

Nota:
• La mappa deve avere esattamente nrows*ncols caratteri.
• L'ordine è: riga 1 (colonne 1..n), riga 2, …

KPMODE

Sintassi:
KPMODE debounce_ms repeat_delay_ms repeat_rate_ms

Descrizione:
Configura i tempi di **debounce**, il **ritardo prima dell'auto-repeat** e la **cadenza dell'auto-repeat** (se si tiene premuto un tasto). Metti repeat_delay_ms = 0 per disabilitare l'auto-repeat.

Esempi pratici

Esempio 1 – Valori consigliati

```
10 KPMODE 30 500 150
RUN
```

Esempio 2 – Nessun auto-repeat

```
10 KPMODE 30 0 0
RUN
```

Esempio 3 – Debounce più aggressivo

```
10 KPMODE 60 500 150
RUN
```

Nota:
• Funziona dopo KPINIT.
• I millisecondi sono interi non negativi.

KPREAD

Sintassi:
KPREAD var$

Descrizione:
Lettura **non bloccante**: mette nella variabile **stringa** var$ il **prossimo tasto** presente nel buffer (stringa vuota "" se non c'è nulla). Compatibile con PRINT, IF, LET.

Esempi pratici

Esempio 1 – Lettura semplice

```
10 KPREAD K$
20 IF K$<>"" THEN PRINT "KEY=";K$
RUN
```

Esempio 2 – Poll continuo

```
10 KPREAD K$
20 IF K$<>"" THEN PRINT K$
30 GOTO 10
RUN
```

Esempio 3 – Uso con mappa

```
10 KPINIT 2 2 13 12 14 27
20 KPMAP "ABCD"
30 KPREAD T$
40 IF T$="A" THEN PRINT "LED ON"
RUN
```

Nota:
• var$ deve terminare con $ (variabile stringa).
• Non si blocca: se non c'è un tasto, ritorna "".

KPWAIT

Sintassi:
KPWAIT var$

Descrizione:
Lettura **bloccante**: attende finché non viene premuto un tasto valido e lo memorizza in var$.

Esempi pratici

Esempio 1 – Attendi e stampa

```
10 PRINT "Premi un tasto..."
20 KPWAIT K$
30 PRINT "Hai premuto: ";K$
RUN
```

Esempio 2 – Menu a 4 tasti

```
10 KPWAIT K$
20 IF K$="A" THEN PRINT "LED ON"
30 IF K$="B" THEN PRINT "LED OFF"
40 IF K$="C" THEN PRINT "START"
50 IF K$="D" THEN PRINT "STOP"
60 GOTO 10
RUN
```

Esempio 3 – Con mappa 4×4

```
10 KPINIT 4 4 2 3 4 5 6 7 8 9
20 KPMAP "123A456B789C*0#D"
30 KPWAIT K$
40 PRINT "KEY=";K$
RUN
```

Nota:
• Richiede KPINIT (e opzionalmente KPMAP).
• Blocca finché non arriva un tasto con debounce applicato.

LCD BACKLIGHT *(solo I²C)*

Sintassi

```
LCD BACKLIGHT ON
LCD BACKLIGHT OFF
```

Descrizione
Accende/spegne la **retroilluminazione** (supportato sui moduli I²C).

Esempio

```
10 LCD ROW 0 "Backlight OFF in 1s"
20 DELAY 1000
30 LCD BACKLIGHT OFF
40 DELAY 1500
50 LCD BACKLIGHT ON
```

Note

- Non disponibile sul parallelo; se chiamato potrebbe dare errore/venire ignorato (dipende dalla tua implementazione).

LCD CLEAR

Sintassi

LCD CLEAR

Descrizione
Cancella tutto lo schermo e azzera la memoria del display.

Esempio

10 LCD CLEAR

Note

- Dopo CLEAR il cursore torna tipicamente a (0,0) (dipende dalla libreria; in ogni caso puoi usare LCD HOME).

LCD CURSOR

Sintassi

LCD CURSOR ON|OFF [BLINK ON|OFF]

Descrizione

Mostra/nasconde il cursore e abilita/disabilita il **lampeggio**.

Esempi

```
10 LCD ROW 0 "Cursor ON, no blink"
20 LCD CURSOR ON BLINK OFF
30 DELAY 1500
40 LCD ROW 1 "Now BLINK ON"
50 LCD CURSOR ON BLINK ON
60 DELAY 1500
70 LCD CURSOR OFF
```

Note

- Mappa a cursor()/noCursor() e blink()/noBlink() della libreria.

LCD HOME

Sintassi

LCD HOME

Descrizione
Riporta il cursore alla posizione (0,0) senza cancellare il contenuto.

Esempio

10 LCD HOME

LCD I2C INIT (16X2 – 16X4)

Sintassi

LCD I2C INIT <addr> [cols rows] [sda scl]

Descrizione

Inizializza un LCD **I²C** (PCF8574).
<addr> accetta esadecimale **senza** prefisso (27, 3F) oppure con 0x (0x27).
Opzionali: dimensioni (cols rows) e pin SDA SCL (per ESP32).

Esempi

```
10 LCD I2C INIT 27 16 2
20 LCD CLEAR
30 LCD ROW 0 "HELLO I2C"
```

Note

- Range indirizzo valido: **03..77 (hex)**.
- Su alcune board i pin I²C **default** vanno già bene; specifica SDA SCL solo se necessario.

LCD PAR INIT (16X2 – 16X4)

Sintassi

LCD PAR INIT <rs> <en> <d4> <d5> <d6> <d7> [cols rows]

Descrizione

Inizializza un LCD **parallelo** compatibile HD44780. Se non specifichi cols rows, usa 16 2.

Esempio

```
10 LCD PAR INIT 12 11 5 4 3 2 16 2
20 LCD CLEAR
30 LCD ROW 0 "HELLO PARALLEL"
```

Note

- Dopo l'init puoi usare tutti i comandi LCD di stampa/controllo.
- Per altri formati, imposta cols rows (es. 16 4).

LCD PRINT

Sintassi

LCD PRINT <testo/expr>

Descrizione
Stampa dalla posizione **corrente**. Accetta stringhe, variabili ed **espressioni** con +.

Esempi

```
10 A$ = "WORLD"
20 N = 42
30 LCD PRINT "HELLO " + A$
40 LCD PRINT "  N=" + N
```

Note

- Non va a capo automatico tra righe: per stampare su un’altra riga usa LCD SETCUR o LCD ROW.

LCD ROW

Sintassi

LCD ROW <row> <testo/expr>

Descrizione

Scrive l'intera **riga** <row> a partire dalla colonna 0.
Il testo viene **riempito con spazi** fino a cols ed **eventualmente troncato** se troppo lungo.

Esempi

```
10 T = 23
20 H = 55
30 LCD ROW 0 "TEMP=" + T + "C"
40 LCD ROW 1 "HUM=" + H + "%"
```

Note

- Ideale per aggiornamenti "puliti" di una riga informativa.

LCD SCROLL

Sintassi

LCD SCROLL LEFT [n]
LCD SCROLL RIGHT [n]

Descrizione

Scorre l’intero display a **sinistra** o **destra** di n passi (default 1).

Esempi

```
10 LCD ROW 0 "Scrolling LEFT >>>"
20 DELAY 1000
30 LCD SCROLL LEFT 5
40 DELAY 1000
50 LCD ROW 1 "<<< Scrolling RIGHT"
60 LCD SCROLL RIGHT 5
```

LCD SETCUR

Sintassi

LCD SETCUR <col> <row>

Descrizione
Posiziona il cursore in **colonna** <col> e **riga** <row>.

Esempio

```
10 LCD SETCUR 2 1
20 LCD PRINT "Qui"
```

Note

- Valori fuori range vengono limitati a [0..cols-1] e [0..rows-1].

LEFT$(A$, N)

Sintassi:

LEFT$(stringa$, N)

Descrizione:

La funzione LEFT$ restituisce una **sottostringa** contenente i **primi N caratteri** della stringa A$.
Se N è maggiore della lunghezza della stringa, viene restituita **l'intera stringa**.

Utile per:

- Analisi o taglio di stringhe
- Parsing di input
- Verifica di prefissi o codici

Esempi pratici

Esempio 1 – Primi 4 caratteri

```
10 A$ = "BASIC32"
20 PRINT LEFT$(A$, 4)
RUN
```

Output atteso:

```
BASI
```

Esempio 2 – Uso in IF per riconoscere un comando

```
10 COM$ = "LOAD file.bas"
20 IF LEFT$(COM$, 4) = "LOAD" THEN PRINT "COMANDO DI CARICAMENTO"
RUN
```

Output atteso:

```
COMANDO DI CARICAMENTO
```

Esempio 3 – Valore di N maggiore della lunghezza

```
10 T$ = "ESP"
20 PRINT LEFT$(T$, 10)
RUN
```

Output atteso:

```
ESP
```

Esempio 4 – Sottostringa con N = 0

```
10 A$ = "TEST"
20 PRINT LEFT$(A$, 0)
RUN
```

Output atteso:

(empty string)

Esempio 5 – Lettura di codice numerico da inizio riga

```
10 RIGA$ = "12345: PRINT 'CIAO'"
20 CODICE$ = LEFT$(RIGA$, 5)
30 PRINT "CODICE = "; CODICE$
RUN
```

Output atteso:

CODICE = 12345

Nota:

- N deve essere ≥ 0
- Funziona solo con variabili stringa ($)
- Combinabile con RIGHT$, MID$, LEN, ASC, CHR$, ecc.

LEN(A$)

Sintassi:

```
LEN(stringa$)
```

Descrizione:
La funzione LEN restituisce la **lunghezza** (in caratteri) di una **stringa**.
Conta **ogni carattere**, inclusi spazi, simboli, numeri, lettere, ecc.

È utile per:

- Verificare input dell'utente
- Controllare se una stringa è vuota
- Lavorare con substringhe

Esempi pratici

Esempio 1 – Lunghezza di una stringa

```
10 A$ = "CIAO"
20 PRINT "LUNGHEZZA: "; LEN(A$)
RUN
```

Output atteso:

```
LUNGHEZZA: 4
```

Esempio 2 – Stringa con spazi

```
10 T$ = "CIAO MONDO"
20 PRINT LEN(T$)
RUN
```

Output atteso:

```
10
```

Esempio 3 – Stringa vuota

```
10 V$ = ""
20 PRINT LEN(V$)
RUN
```

Output atteso:

```
0
```

Esempio 4 – Uso in condizione IF

```
10 INPUT "INSERISCI TESTO: "; T$
20 IF LEN(T$) = 0 THEN PRINT "NESSUN DATO INSERITO"
RUN
```

Output atteso (se si preme solo INVIO):

```
NESSUN DATO INSERITO
```

Esempio 5 – Contatore caratteri

```
10 MSG$ = "BASIC32"
20 PRINT "NUMERO DI CARATTERI: "; LEN(MSG$)
RUN
```

Output atteso:

```
NUMERO DI CARATTERI: 7
```

Nota:

- LEN funziona **solo con variabili stringa** ($)
- Non restituisce errori se la stringa è vuota: ritorna 0
- Combinabile con LEFT$, RIGHT$, MID$, CHR$, ASC

LET

Sintassi:

```
LET variabile = espressione
```

oppure semplicemente:

```
variabile = espressione
```

Descrizione:
Il comando LET serve per **assegnare un valore a una variabile**, sia numerica che stringa ($).
È **opzionale**: puoi ometterlo e scrivere direttamente l'assegnazione (come nei BASIC più moderni).

Può assegnare:

- costanti numeriche
- stringhe
- risultati di espressioni o funzioni
- espressioni condizionali

Esempi pratici

Esempio 1 – Assegnazione numerica semplice

```
10 LET A = 5
20 PRINT A
RUN
```

Output atteso:

```
5
```

Esempio 2 – Uso senza LET (forma abbreviata)

```
10 B = 10 * 2
20 PRINT B
RUN
```

Output atteso:

```
20
```

Esempio 3 – Assegnazione stringa

```
10 LET NOME$ = "LUCA"
```

```
20 PRINT "CIAO "; NOME$
RUN
```

Output atteso:

```
CIAO LUCA
```

Esempio 4 – Con funzioni

→ Assegnare a una variabile il risultato di una funzione:

```
10 X = SQR(49)
20 PRINT "RADICE: "; X
RUN
```

Output atteso:

```
RADICE: 7
```

Esempio 5 – Assegnazione condizionale

→ Usa IF per assegnare valori diversi:

```
10 A = 3
20 IF A < 5 THEN LET RISULTATO = 1 ELSE LET RISULTATO = 0
30 PRINT RISULTATO
RUN
```

Output atteso:

```
1
```

Nota:

- Non è possibile assegnare array direttamente (A(1) = ...) se non prima di un DIM
- Puoi usare LET per migliorare la leggibilità, anche se non è obbligatorio

LIST

Sintassi

LIST
LIST start
LIST start-end
LIST -end
LIST start-
LIST "nomefile.bas"
LIST "nomefile.bas" start
LIST "nomefile.bas" start-end
LIST "nomefile.bas" -end
LIST "nomefile.bas" start-

Descrizione

Il comando **LIST** visualizza:

1. Il programma attualmente in memoria

Se non viene specificato un nome file, LIST mostra il listato BASIC presente in RAM, ordinato per numero di riga.

Le righe generate da INCLUDE non vengono mostrate.

2. Il contenuto di un file .bas senza caricarlo

Se viene specificato un nome file tra virgolette, il comando visualizza il contenuto del file da SPIFFS (o SD se esteso), senza modificarne il programma attuale.

In entrambi i casi è possibile limitare l'output con intervalli di righe.

Esempi

Visualizzare tutto il programma in memoria

LIST

Visualizzare dalla riga 200 in poi

LIST 200

Visualizzare le righe da 200 a 300

LIST 200-300

Visualizzare dall'inizio fino a riga 300

LIST -300

Visualizzare dalla riga 500 fino alla fine

LIST 500-

Visualizzare il contenuto di un file

LIST "modulo.bas"

Output di esempio:

```
10 FUNC TEST
20 PRINT "ok"
30 ENDFUNC
```

Visualizzare un file applicando un intervallo di righe

LIST "modulo.bas" 200-
LIST "modulo.bas" -300
LIST "modulo.bas" 100-200

Note

- LIST senza virgolette mostra il programma in RAM.
- LIST "file.bas" mostra il contenuto del file **senza caricarlo**.
- Le righe da INCLUDE non vengono mostrate nel listato del programma.
- Per cancellare tutto il programma usare NEW.
- Le righe si modificano riscrivendo lo stesso numero di riga.

LISTVARS

Nome

LISTVARS

Sintassi

LISTVARS "file"
LISTVARS SPIFFS "file"
LISTVARS SD "file"

Descrizione

LISTVARS stampa a video tutte le coppie **chiave = valore** presenti nel file JSON.

È utile per:

- debug
- verifica del contenuto del file
- controllo rapido dopo SAVEVAR e DELVAR

Esempio – Visualizzare variabili salvate su SD

10 SAVEVAR SD "dati.json" "TEMP" 22.5
20 SAVEVAR SD "dati.json" "STATO" "OK"
30 LISTVARS SD "dati.json"

Output atteso:

Listing variables in /dati.json:
  TEMP = 22.5
  STATO = "OK"

Note

- Se il file non esiste, l'elenco risulta vuoto (o viene creato un oggetto vuoto internamente).
- Le stringhe vengono mostrate tra virgolette.
- L'ordine delle chiavi può non essere sempre lo stesso (dipende dal JSON).

LOAD

(o LOAD F$)

Sintassi

LOAD "nomefile"
LOAD variabile$
LOAD SD "nomefile"
LOAD SD variabile$
LOAD SPIFFS "nomefile"
LOAD SPIFFS variabile$

Descrizione

Il comando **LOAD** carica un file .bas nella memoria programma, sovrascrivendo completamente il listato attuale.

- Se non viene specificata alcuna memoria, LOAD legge il file dalla memoria interna **SPIFFS**.
- Indicando **SD**, il file viene caricato dalla **scheda SD**.
- È possibile indicare **SPIFFS** anche esplicitamente, ma non è obbligatorio.
- Il nome del file può essere specificato direttamente tra virgolette oppure tramite una **variabile stringa** (es. F$).

Il caricamento sostituisce completamente il programma BASIC attualmente in memoria.

Comportamento in un listato BASIC

Quando LOAD viene eseguito **all'interno di un programma BASIC** (cioè con un numero di linea), il comportamento cambia:

Il comando:

1. **Interrompe immediatamente** il programma in corso.
2. **Carica** il nuovo file BASIC indicato.
3. **Avvia automaticamente** il nuovo programma, come se fosse stato eseguito con RUN "nomefile".

In pratica, LOAD all'interno di un listato agisce come un **"CHAIN & RUN"**, sostituendo il programma in memoria e lanciandolo subito.

Esempi pratici

Esempio 1 – Caricare un file dalla memoria interna (SPIFFS)

```
LOAD "main.bas"
```

Output atteso:
Il listato presente in main.bas viene caricato dalla memoria interna SPIFFS.

Esempio 2 – Caricare un file da scheda SD

```
LOAD SD "menu.bas"
```

Output atteso:
Il listato presente in menu.bas viene caricato dalla scheda SD.

Esempio 3 – Caricare da variabile stringa

```
10 LET F$ = "config.bas"
20 LOAD F$
RUN
```

Output atteso:
Il programma config.bas viene caricato dalla memoria interna SPIFFS.

Esempio 4 – Caricare da SD con variabile

```
10 LET F$ = "gioco.bas"
20 LOAD SD F$
RUN
```

Output atteso:
Il programma gioco.bas viene caricato dalla scheda SD.

Esempio 5 – Caricare ed eseguire un altro programma da un listato

```
10 PRINT "Avvio loader..."
20 LOAD "main.bas"
```

Output atteso:
Il programma main.bas viene caricato, il programma corrente si interrompe,
e main.bas inizia automaticamente l'esecuzione come se fosse stato lanciato con RUN.

Note

- Se la scheda SD non è presente o non è montata, LOAD SD restituirà un errore.

- Se il file non esiste, viene mostrato un messaggio di errore (“File not found” o “Loading failed”).
- È possibile usare variabili stringa per specificare dinamicamente il nome del file.
- Se LOAD viene eseguito da un listato BASIC, **non è necessario** aggiungere RUN dopo: l’esecuzione parte automaticamente.

LOADVAR

Nome

LOADVAR

Sintassi

LOADVAR "file" "chiave" variabile
LOADVAR SPIFFS "file" "chiave" variabile
LOADVAR SD "file" "chiave" variabile

Dove variabile può essere:

- Numerica: A, S, TEMP, SPEED
- Stringa: A$, U$, NOME$, USER$

Descrizione

LOADVAR legge una chiave da un file JSON e mette il valore in una variabile BASIC.

- Se la variabile di destinazione termina con $ → viene trattata come **stringa**
- Altrimenti → viene trattata come **numero**

Esempi

Esempio 1 – Caricare impostazioni da SPIFFS

```
10 LOADVAR SPIFFS "prefs.json" "SPEED" S
20 LOADVAR SPIFFS "prefs.json" "USER" U$
30 PRINT S
40 PRINT U$
```

Output atteso:

```
150
Anna
```

Esempio 2 – Caricare da SD

```
10 LOADVAR SD "dati.json" "TEMP" T
20 PRINT "TEMP="; T
```

Note

- Se la chiave non esiste nel file → errore runtime (chiave non trovata).
- Se carichi un numero dentro una variabile stringa (A$) il valore viene convertito in testo.
- Se carichi una stringa dentro una variabile numerica (A) il runtime prova a convertirla in numero (se non convertibile, spesso diventa 0).

LOG(x)

Sintassi:

LOG(x)

Descrizione:
La funzione LOG(x) restituisce il **logaritmo naturale** (in base **e**) di un numero positivo x.
La base e (circa **2.71828**) è la base dei logaritmi naturali comunemente usati in matematica e fisica.

- LOG(1) = 0
- LOG(e) = 1
- LOG(x) è **definito solo per x > 0**

 Se x è zero o negativo, il risultato non è valido e può generare errore o valore indefinito.

Esempi pratici

Esempio 1 – Logaritmo naturale di 1

```
10 PRINT "LOG(1) = "; LOG(1)
RUN
```

Output atteso:

```
LOG(1) = 0
```

Esempio 2 – Logaritmo di e (circa 2.71828)

```
10 PRINT "LOG(E) = "; LOG(2.71828)
RUN
```

Output atteso:

```
LOG(E) = 1
```

Esempio 3 – Logaritmo di un valore maggiore

```
10 X = 10
20 PRINT "LOG(10) = "; LOG(X)
RUN
```

Output atteso:

```
LOG(10) = 2.30258
```

Esempio 4 – Uso in formula combinata
→ Calcolo della funzione f(x) = LOG(x) * x

```
10 INPUT A
20 PRINT "f(A) = "; LOG(A) * A
RUN
```

Output atteso (es. input 5):

```
f(A) = 8.047
```

Nota:

- Per calcolare logaritmi in base 10, puoi usare:

  LOG10(X) = LOG(X) / LOG(10)

MAC

Sintassi

```
MAC      ' funzione numerica
MAC$     ' funzione stringa
```

Descrizione

Il comando **MAC** restituisce il MAC address dell'ESP32 associato all'interfaccia Wi-Fi.

- MAC → restituisce il MAC come **numero** (48 bit in decimale)
- MAC$ → restituisce il MAC come **stringa** nel formato AA:BB:CC:DD:EE:FF
- Quando assegni a una variabile **stringa**, MAC viene automaticamente trattato come MAC$:
- A$ = MAC   ' equivale a A$ = MAC$

Come per il comando IP, è consigliato usare MAC **dopo il comando WIFI**, così il Wi-Fi è inizializzato correttamente.
Se il Wi-Fi non è stato inizializzato, il valore può risultare 0 / 00:00:00:00:00:00.

Esempi pratici

Esempio 1 – Connessione Wi-Fi e stampa del MAC in formato stringa

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 PRINT MAC$
RUN
```

Output atteso (esempio):

```
AA:BB:CC:DD:EE:FF
```

Esempio 2 – Assegnazione numerica e stringa

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 M = MAC
40 M$ = MAC$
50 PRINT "MAC NUMERICO ="; M
60 PRINT "MAC STRINGA ="; M$
RUN
```

Output atteso (esempio):

MAC NUMERICO = 187723572702975
MAC STRINGA = AA:BB:CC:DD:EE:FF

Esempio 3 – Assegnazione con scorciatoia

```
10 WIFI "ssid" "password"
20 WAIT 3000
30 A$ = MAC
40 PRINT "IL MIO MAC È:"; A$
RUN
```

Output atteso:

IL MIO MAC È: AA:BB:CC:DD:EE:FF

(grazie alla scorciatoia, A$ = MAC viene interpretato come A$ = MAC$)

Nota

- Richiede che il sottosistema Wi-Fi sia inizializzato (ad es. via WIFI "ssid" "password"), altrimenti può restituire:
  - MAC = 0
  - MAC$ = "00:00:00:00:00:00"
- MAC è utile in forma numerica per identificatori, hashing, confronti.
- MAC$ è ideale per log, debug, stampe su seriale e file.
- Compatibile con PRINT, LET, IF, variabili numeriche e stringa.

MEMCLEAN

Sintassi:
MEMCLEAN <tipo>
Dove <tipo> può essere:
STRING, VARIABLE, ARRAY, HTML, GRAPHICS, STACK, ALL

Descrizione:
Il comando MEMCLEAN libera porzioni specifiche di memoria rimuovendo i dati memorizzati in runtime.
È utile per ottimizzare l'uso della RAM, prevenire rallentamenti o blocchi durante esecuzioni cicliche o animate.
Può essere chiamato anche più volte durante il programma.

Esempi pratici

Esempio 1 – Pulire le variabili stringa

```
10 MEMCLEAN STRING
RUN
```

Output atteso:
Tutte le variabili stringa ($) vengono cancellate dalla memoria.

Esempio 2 – Pulire tutte le variabili numeriche

```
10 MEMCLEAN VARIABLE
RUN
```

Output atteso:
Tutte le variabili numeriche vengono azzerate (non più definite).

Esempio 3 – Pulire array definiti

```
10 MEMCLEAN ARRAY
RUN
```

Output atteso:
Tutti gli array allocati vengono liberati.

Esempio 4 – Rimuovere contenuto HTML dalla memoria

```
10 MEMCLEAN HTML
RUN
```

Output atteso:
Viene cancellata ogni pagina HTML memorizzata o bufferizzata.

Esempio 5 – Pulire grafica e sprite

```
10 MEMCLEAN GRAPHICS
RUN
```

Output atteso:
Tutti gli sprite vengono disattivati.
Le mappe grafiche (OLEDDATA, ILIDATA) vengono eliminate.
Il display OLED/ILI viene pulito.

Esempio 6 – Pulire lo stack dei GOSUB

```
10 MEMCLEAN STACK
RUN
```

Output atteso:
Lo stack dei salti GOSUB viene svuotato. Utile se si sospetta overflow o ricorsioni interrotte.

Esempio 7 – Pulizia completa di tutta la memoria utente

```
10 MEMCLEAN ALL
RUN
```

Output atteso:
Tutti i dati in memoria vengono azzerati: stringhe, variabili, array, HTML, grafica e stack.

Nota:

- I parametri sono parole chiave **senza virgolette**: STRING, VARIABLE, ARRAY, HTML, GRAPHICS, STACK, ALL
- La chiamata è **case-insensitive** (memclean(graphics) è valido)
- Può essere usato per recuperare RAM prima o dopo cicli animati
- Utile per programmi lunghi o dinamici con risorse grafiche o stack GOSUB

MID$(A$, start, len)

Sintassi:

MID$(stringa$, inizio, lunghezza)

Descrizione:
La funzione MID$ estrae una **sottostringa** da A$ a partire dalla posizione inizio (1 = primo carattere), lunga al massimo lunghezza caratteri.
Se inizio supera la lunghezza della stringa, il risultato è vuoto.
Se inizio + len supera la lunghezza della stringa, viene estratta solo la parte disponibile.

Esempi pratici

Esempio 1 – Estrai "SIC" da "BASIC32"

```
10 A$ = "BASIC32"
20 PRINT MID$(A$, 3, 3)
RUN
```

Output atteso:

```
SIC
```

Esempio 2 – Estrai l'estensione da un file

```
10 F$ = "SETUP.BAS"
20 PRINT MID$(F$, 6, 4)
RUN
```

Output atteso:

```
.BAS
```

Esempio 3 – Estrai una sola lettera

```
10 S$ = "HELLO"
20 PRINT MID$(S$, 2, 1)
RUN
```

Output atteso:

```
E
```

Esempio 4 – Indice oltre la lunghezza

```
10 X$ = "TEST"
```

```
20 PRINT MID$(X$, 10, 2)
RUN
```

Output atteso:

(empty string)

Esempio 5 – Uso con variabili

```
10 T$ = "BENVENUTO"
20 INIZIO = 4
30 LUN = 5
40 PRINT MID$(T$, INIZIO, LUN)
RUN
```

Output atteso:

```
VENUT
```

Nota:

- L’indice parte da **1**, non da 0
- La lunghezza specificata può eccedere la fine della stringa: l’output sarà comunque valido
- Combinabile con LEFT$, RIGHT$, LEN, ASC, CHR$, ecc.

MILLIS

Sintassi:

MILLIS n

Descrizione:

Il comando **MILLIS** sospende **solo l'esecuzione del programma BASIC** per *n* millisecondi,
ma **non blocca il microcontrollore**.

Durante l'attesa, l'ESP32 continua ad eseguire tutte le altre operazioni di sistema:

- server web
- WiFi / MQTT
- input da tastiera o seriale
- funzioni BASIC in background (FUNC ... LOOP)
- task periodici o altre attività interne

È quindi un **ritardo cooperativo**: il programma BASIC si "ferma" momentaneamente, ma il sistema continua a funzionare normalmente.

È utile per:

- Creare **animazioni o lampeggi** senza bloccare il sistema
- Gestire **ritardi temporali** mentre altre funzioni continuano a lavorare
- Sincronizzare più routine BASIC che girano in parallelo
- Sostituire DELAY in modo più efficiente

Differenza tra DELAY e MILLIS

Comando	Tipo di pausa	Il microcontrollore continua a lavorare?
DELAY n	Bloccante	□ No, tutto si ferma
MILLIS n	Non bloccante (cooperativa)	□ Sì, continua a lavorare

---

Esempi pratici

Esempio 1 – Pausa non bloccante tra due messaggi

```
10 PRINT "CIAO"
20 MILLIS 1000
30 PRINT "MONDO"
```

RUN

Output atteso:

CIAO
(paura di 1 secondo)
MONDO

Durante la pausa di 1 secondo, il microcontrollore resta attivo (server web, MQTT, ecc.).

Esempio 2 – Lampeggio senza bloccare il sistema

```
10 OUT 2, 1       ' Accende LED
20 MILLIS 500
30 OUT 2, 0       ' Spegne LED
40 MILLIS 500
50 GOTO 10
RUN
```

Output atteso:
Il LED su GPIO2 lampeggia ogni mezzo secondo,
mentre il resto del sistema (WiFi, funzioni BASIC in background, ecc.) continua a funzionare.

Esempio 3 – Conto alla rovescia con altre attività attive

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 MILLIS 1000
40 NEXT I
50 PRINT "VIA!"
RUN
```

Output atteso:

5
4
3
2
1
VIA!

Il countdown si svolge in tempo reale, ma il sistema non resta bloccato.

Esempio 4 – Due processi in parallelo (multitasking cooperativo)

```
10 FUNC BLINK
20 OUT 2,1
30 MILLIS 250
40 OUT 2,0
50 MILLIS 250
60 GOTO 20
70 ENDFUNC

80 FUNC CLOCK
90 PRINT "Millis:", SYS.MILLIS
100 MILLIS 1000
110 GOTO 90
120 ENDFUNC

130 STARTFUNC BLINK
140 STARTFUNC CLOCK
150 END
```

Effetto:

- Il LED lampeggia ogni 250 ms
- Il tempo (in millisecondi) viene stampato ogni secondo
- Entrambe le funzioni girano *in parallelo*, senza bloccare nulla

---

Esempio 5 – Simulazione di caricamento cooperativo

```
10 PRINT "CARICAMENTO"
20 FOR I=1 TO 10
30 PRINT ".";
40 MILLIS 300
50 NEXT I
60 PRINT "COMPLETATO!"
RUN
```

Output atteso:

```
CARICAMENTO.....
COMPLETATO!
```

Durante i 3 secondi complessivi di attesa, l’ESP32 resta operativo.

Note:

- **MILLIS** misura il tempo in millisecondi (1000 = 1 secondo)
- Non blocca input, rete o altre funzioni BASIC
- Può essere usato all'interno di funzioni (FUNC) o cicli
- Se usato nella console interattiva (fuori da RUN), si comporta come un DELAY tradizionale
- È ideale per programmi multitasking o con server attivi

MQTTAUTOPOLL

Sintassi:
MQTTAUTOPOLL ON <intervallo_ms>
MQTTAUTOPOLL OFF

Descrizione:
Il comando MQTTAUTOPOLL abilita o disabilita il polling automatico del client MQTT. Quando attivo, il dispositivo controlla periodicamente se sono arrivati nuovi messaggi MQTT.
È necessario per ricevere messaggi in tempo reale senza bloccare l'esecuzione del programma.

Esempi pratici

Esempio 1 – Abilitare il polling ogni 100 ms

```
10 MQTTCONNECT "192.168.1.49" 1883 "" ""
20 MQTTSUB "casa/comandi"
30 MQTTAUTOPOLL ON, 100
RUN
```

Output atteso:
Il dispositivo controlla ogni 100 millisecondi se ci sono nuovi messaggi sul topic casa/comandi.

Esempio 2 – Disattivare il polling automatico

```
10 MQTTAUTOPOLL OFF
RUN
```

Output atteso:
Il dispositivo smette di controllare automaticamente i messaggi MQTT.

Nota:
• Il polling è fondamentale per la ricezione automatica dei messaggi
• Il parametro <intervallo_ms> è l'intervallo in millisecondi tra ogni controllo
• Usa OFF per disattivare completamente il polling
• Va attivato solo dopo MQTTCONNECT e MQTTSUB

MQTTCONNECT

Sintassi:
MQTTCONNECT "broker" porta "user" "password"

Descrizione:
Il comando MQTTCONNECT permette di connettere il dispositivo a un broker MQTT (come Mosquitto o un broker cloud) specificando l'indirizzo, la porta e le eventuali credenziali di accesso.
Una volta connesso, il dispositivo può pubblicare e ricevere messaggi MQTT.

Esempi pratici

Esempio 1 – Connessione senza autenticazione

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "192.168.1.100" 1883 "" ""
RUN
```

Output atteso:
Connessione stabilita con il broker MQTT locale all'indirizzo 192.168.1.100.

Esempio 2 – Connessione a un broker con credenziali

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "mqtt.mioserver.com" 1883 "utente1" "passw1"
RUN
```

Output atteso:
Connessione al broker mqtt.mioserver.com sulla porta 1883 usando nome utente e password.

Nota:
• La porta standard MQTT è **1883**
• Se il broker non richiede autenticazione, usare "" per user e password
• Richiede una connessione Wi-Fi attiva (WIFI)
• Deve essere seguito da MQTTSUB, MQTTPUBLISH, ecc.
• Se la connessione fallisce, viene segnalato nel monitor seriale
• Non è compatibile con MQTT over TLS (porta 8883)

MQTTPUB

Sintassi:
MQTTPUB "<topic>" "<messaggio>"

Descrizione:
Il comando MQTTPUB pubblica un messaggio sul topic MQTT specificato.
Permette di inviare comandi o notifiche a sistemi esterni, come altri dispositivi o server domotici.

Esempi pratici

Esempio 1 – Inviare un messaggio a un topic

```
10 MQTTPUB "casa/luci" "ON"
RUN
```

Output atteso:
Il messaggio "ON" viene pubblicato sul topic casa/luci.

Esempio 2 – Inviare il contenuto di una variabile

```
10 LET M$ = "Temperatura OK"
20 MQTTPUB "sistema/stato" M$
RUN
```

Output atteso:
Pubblica il contenuto della variabile M$ sul topic sistema/stato.

Nota:
• Entrambi i parametri devono essere stringhe
• È necessario aver eseguito prima MQTTCONNECT
• Può essere usato in risposta a eventi o comandi ricevuti
• Utile per dialogare con Home Assistant, Node-RED, ESP32 remoti, ecc.

MQTTSUB

Sintassi:
MQTTSUB "<topic>"

Descrizione:
Il comando MQTTSUB sottoscrive il dispositivo a un topic MQTT specifico.
Dopo la sottoscrizione, ogni messaggio ricevuto da quel topic sarà automaticamente salvato nella variabile MSG$.
Può essere gestito in tempo reale tramite DO o controllato manualmente.

Esempi pratici

Esempio 1 – Iscriversi a un topic e stampare i messaggi

```
10 WIFI "ssid" "password"
20 MQTTCONNECT "192.168.1.49" 1883 "" ""
30 MQTTSUB "sistema/stato"
40 MQTTAUTOPOLL ON 100
50 DO 100
100 IF MSG$ <> "" THEN PRINT MSG$
110 LET MSG$ = ""
RUN
```

Output atteso:
La variabile MSG$ prende il valore ricevuto dal topi MQTT.

Esempio 2 – Accendere e spegnere un LED da MQTT

```
10 PINMODE 2, OUTPUT NOPULL
20 WIFI "ssid" "password"
30 MQTTCONNECT "192.168.1.49" 1883 "" ""
40 MQTTSUB "casa/comandi"
50 MQTTAUTOPOLL ON 100
60 DO 100
70 DO 110
100 IF MSG$ = "ON" THEN DWRITE 2 1
110 IF MSG$ = "OFF" THEN DWRITE 2 0
RUN
```

Output atteso:
Il LED sul pin 2 si accende o si spegne in base ai messaggi ricevuti (ON / OFF) sul topic casa/comandi.

Nota:
• Il topic va racchiuso tra virgolette
• Necessaria connessione MQTT (MQTTCONNECT)

- MSG$ contiene il messaggio ricevuto più recente
- Usare MQTTAUTOPOLL per ricezione continua
- Può essere combinato con DO o IF

NEW

Sintassi:

NEW

Descrizione:
Il comando NEW cancella **tutto il programma attualmente in memoria**, liberando spazio per scrivere un nuovo listato.
Dopo l'esecuzione, la memoria programma è **vuota**, ma le variabili restano definite fino a un nuovo RUN o CLR.

Non chiede conferma: appena eseguito, elimina tutto il codice presente.

Esempi pratici

Esempio – Azzerare il programma corrente

NEW

Output atteso:
Il listato in memoria viene cancellato. Nessuna riga viene mostrata con LIST.

Nota:

- NEW **non cancella i file salvati**, solo il contenuto della RAM.

NOTONE

Sintassi:

NOTONE pin

Descrizione:

Ferma il tono in corso sul pin specificato, utile se si vuole interrompere un suono emesso con TONE.

Esempi pratici

```
10 TONE 26 1000 10000
20 WAIT 2000
30 NOTONE 26
```

Avvia un suono per 10 secondi, ma lo interrompe dopo 2.

Note:

- Se il TONE ha già una durata breve, NOTONE non è necessario.
- Utile per creare effetti sonori controllati da eventi esterni o condizioni logiche.

NOWCLR

Sintassi

NOWCLR

Descrizione

NOWCLR cancella il contenuto della variabile di ricezione impostata da:

NOWINIT recVar$

Quindi, se hai fatto:

NOWINIT RX$

allora NOWCLR azzera **solo** RX$ (o il nome che hai usato in NOWINIT).

Non modifica la coda dei messaggi usata da NOWPOLL, agisce **solo** sulla variabile "storica".

Esempi

Esempio 1 – Ricezione semplice senza coda

```
10 NOWINIT RX$
20 PRINT "RICEVITORE ESP-NOW"
30 PRINT "MAC locale: "; MAC$
40 GOTO 100

100 IF RX$ <> "" THEN PRINT "Ricevuto: "; RX$
110 IF RX$ <> "" THEN NOWCLR
120 GOTO 100
```

Qui:

- RX$ contiene l'ultimo messaggio ricevuto
- dopo averlo stampato viene cancellato con NOWCLR, così non viene elaborato due volte

---

Note

- NOWCLR è utile nella modalità semplice basata su recVar$.
- Se usi NOWPOLL, non è necessario chiamare NOWCLR: la coda viene gestita da NOWPOLL e il messaggio estratto viene già "consumato".

- Se NOWINIT non è mai stato chiamato, NOWCLR non fa nulla.

NOWINIT (ESP-NOW)

Sintassi

NOWINIT recVar$

- recVar$ → variabile stringa che conterrà **l'ultimo messaggio ricevuto** via ESP-NOW
  (usata anche da NOWCLR)

Descrizione

NOWINIT inizializza il supporto **ESP-NOW** nel tuo interprete:

- imposta il Wi-Fi in modalità **WIFI_STA**
- inizializza il protocollo ESP-NOW
- registra la variabile stringa recVar$ come **buffer di ricezione "storico"**
- svuota la **coda interna** dei messaggi usata da NOWPOLL

Deve essere eseguito **prima di usare**:

- NOWSEND
- NOWCLR
- NOWPOLL

Se l'inizializzazione fallisce, genera un errore di runtime:

ESP-NOW initialization error.

Esempi

Esempio 1 – Inizializzazione di un ricevitore

```
10 NOWINIT RX$
20 PRINT "RICEVITORE ESP-NOW"
30 PRINT "MAC locale: "; MAC$
```

Esempio 2 – Inizializzazione di un trasmettitore

```
10 NOWINIT TX$
20 PRINT "TRASMETTITORE ESP-NOW"
30 PRINT "MAC locale: "; MAC$
```

Note

- Va chiamato **una volta all'inizio** del programma (o dopo NEW).
- Reinizializzare con NOWINIT resetta la coda interna di NOWPOLL.

- recVar$ viene aggiornata con **l’ultimo messaggio ricevuto**, ma la gestione robusta dei messaggi multipli è demandata a NOWPOLL.

NOWPOLL

Sintassi

NOWPOLL msgVar$ numVar

- msgVar$ → variabile stringa in cui viene copiato **un messaggio** estratto dalla coda
- numVar → variabile numerica che indica **quanti messaggi restano in coda** dopo il prelievo

---

Descrizione

NOWPOLL legge la **coda interna** dei messaggi ricevuti via ESP-NOW.

La coda:

- è gestita dal firmware (non la vedi direttamente)
- ha capacità massima (es. 8 messaggi)
- viene riempita dalla callback di ricezione ogni volta che arriva un messaggio

Ad ogni chiamata:

- se la coda **contiene almeno un messaggio**:
  - msgVar$ riceve il messaggio più vecchio (FIFO)
  - quel messaggio viene rimosso dalla coda
  - numVar riceve il numero di messaggi **ancora presenti** in coda
- se la coda è **vuota**:
  - msgVar$ viene impostata a ""
  - numVar viene impostata a 0

NOWPOLL è il metodo **consigliato** per ricezione affidabile, soprattutto se il trasmettitore invia rapidamente o a raffica.

Esempi

Esempio 1 – Ricevitore semplice con coda (funzionante con il TX che hai già)

```
10 NOWINIT RX$
20 PRINT "RICEVITORE ESP-NOW"
30 PRINT "MAC: "; MAC$
40 GOTO 100
100 NOWPOLL M$, N
110 IF M$ <> "" THEN PRINT M$, N
120 GOTO 100
```

Con il trasmettitore:

```
10 NOWINIT TX$
20 PRINT "TRASMETTITORE ESP-NOW"
30 PRINT "MAC: "; MAC$
40 DEST$ = "14:33:5C:0E:9A:B8"
50 N = 0
60 GOTO 100
100 N = N + 1
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 200
140 GOTO 100
```

Vedrai sul ricevitore:

```
MSG 1   0
MSG 2   0
MSG 3   0
...
```

Perché il ricevitore è molto veloce e svuota la coda man mano che i messaggi arrivano, quindi **non si accumulano**.

Esempio 2 – Test per vedere N > 0 (coda che si riempie)

Trasmettitore veloce:

```
10 NOWINIT TX$
20 DEST$ = "14:33:5C:0E:9A:B8"
30 N = 0
40 GOTO 100
100 N = N + 1
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 10
140 GOTO 100
```

Ricevitore più lento:

```
10 NOWINIT RX$
20 GOTO 100
100 NOWPOLL M$, N
110 IF M$ <> "" THEN PRINT M$, N
120 WAIT 500
130 GOTO 100
```

In questo caso potrai vedere, ad esempio:

```
MSG 10   3
MSG 11   2
```

MSG 12 1
MSG 13 0

Qui:

- M$ è il messaggio estratto
- N dice quanti messaggi **rimangono ancora in coda** dopo aver letto M$.

Note

- NOWPOLL funziona solo dopo NOWINIT recVar$.
- numVar **non** è la dimensione massima della coda, ma il numero di messaggi **attualmente accodati dopo il POP**.
- Se non ti interessa il conteggio, puoi fare:
- NOWPOLL M$, N
- IF M$ <> "" THEN PRINT M$

  e ignorare N.

- La coda viene svuotata da:
  - NOWINIT (reset)
  - il POP di NOWPOLL (uno alla volta)
  - NEW (reset generale del programma)

NOWSEND

Sintassi

NOWSEND "MAC" msg$
NOWSEND macVar$ msg$

- "MAC" → stringa con MAC del destinatario nel formato "XX:XX:XX:XX:XX:XX"
- macVar$ → variabile stringa che contiene il MAC nello stesso formato
- msg$ → stringa da inviare, può essere:
  - stringa costante "CIAO"
  - variabile stringa (M$, TX$, MSG$, …)

Descrizione

NOWSEND invia un messaggio stringa a un altro dispositivo tramite **ESP-NOW**.

Richiede che NOWINIT sia stato chiamato prima.

- Il MAC può essere:
  - scritto direttamente come stringa: "24:62:AB:F2:4D:CC"
  - messo in una variabile stringa: DEST$ = "24:62:AB:F2:4D:CC"
- Il messaggio può essere costante o in variabile.

Se il MAC non è valido, viene generato un errore di sintassi:

?SYNTAX ERROR
Invalid MAC in NOWSEND

Se l'invio fallisce a runtime:

NOWSEND peer error.

oppure

NOWSEND send failed.

Esempi

Esempio 1 – Trasmettitore continuo (quello che hai usato tu)

```
10 NOWINIT TX$
20 PRINT "TRASMETTITORE ESP-NOW"
30 PRINT "MAC: "; MAC$
40 DEST$ = "14:33:5C:0E:9A:B8"
50 N = 0
60 GOTO 100
100 N = N + 1
```

```
110 TX$ = "MSG " + STR$(N)
120 NOWSEND DEST$ TX$
130 WAIT 200
140 GOTO 100
```

- Il dispositivo invia MSG 1, MSG 2, MSG 3, … ogni 200 ms.
- DEST$ contiene il MAC del ricevitore.

Esempio 2 – Invio singolo con messaggio da variabile

```
10 NOWINIT TX$
20 DEST$ = "24:62:AB:F2:4D:CC"
30 M$ = "CIAO ESP-NOW!"
40 NOWSEND DEST$ M$
```

Note

- NOWINIT deve essere stato eseguito prima di NOWSEND.
- È buona pratica inserire sempre un piccolo WAIT tra due NOWSEND per non saturare la radio.
- Se il destinatario non ha a sua volta inizializzato ESP-NOW, il messaggio può andare perso senza notifica.

NRF AVAILABLE

Sintassi:

NRF AVAILABLE var

Descrizione:

Imposta var a **1** se c'è almeno un payload in ricezione (FIFO RX non vuota), altrimenti **0**. Non consuma il messaggio.

Esempi:

```
10 NRF AVAILABLE A
20 IF A=1 THEN PRINT "C'è un messaggio!"
10 NRF START RX
20 NRF AVAILABLE A
30 IF A=0 THEN WAIT 50 : GOTO 20
40 NRF READ M$
50 PRINT "RX: ";M$
```

Note:

- Usa NRF READ per leggere effettivamente il payload.
- In **RX** continuo, conviene pollare con un piccolo WAIT (20–50 ms).

NRF CONFIG

Sintassi:

```
NRF CONFIG CHANNEL ch
NRF CONFIG DATARATE 250K|1M|2M
NRF CONFIG POWER   MIN|LOW|HIGH|MAX
NRF CONFIG AUTACK  0|1
NRF CONFIG DYNPL   0|1
NRF CONFIG CRCLEN  0|8|16
```

Descrizione:

Modifica la configurazione radio **dopo** NRF INIT.

- CHANNEL (0..125) → frequenza 2.4 GHz + ch*1 MHz.
- DATARATE → velocità (250 Kbps = più sensibile/portata).
- POWER → livello PA (MAX con moduli PA/LNA solo se ben alimentati).
- AUTACK → Auto-Ack/Auto-Retry (abilitato di default).
- DYNPL → payload dinamico (consigliato ON).
- CRCLEN → CRC disattivo/8/16 bit.

Esempi:

```
10 NRF CONFIG DATARATE 250K
20 NRF CONFIG POWER MAX
30 NRF CONFIG CHANNEL 76
```

Note:

- Dopo il cambio, la libreria applica subito i parametri.
- CRCLEN 0 disabilita il CRC (sconsigliato in ambienti rumorosi).

NRF FLUSH

Sintassi:

NRF FLUSH

Descrizione:
Svuota le code **RX** e **TX** del modulo (cancella eventuali payload pendenti).

Esempio:

```
10 NRF FLUSH
20 PRINT "Code pulite"
```

Note:

- Utile dopo errori o cambi di ruolo TX/RX.

NRF INIT (nRF24L01)

Sintassi:

NRF INIT ce csn [channel] [250K|1M|2M] [MIN|LOW|HIGH|MAX] [addrWidth]

Descrizione:
Inizializza il nRF24L01.

- ce, csn: GPIO per CE e CSN (SS/CS del nRF).
- channel: 0..125 (default 76 → 2.476 GHz).
- datarate: default 1M.
- power: default LOW.
- addrWidth: 3..5 (default 5).

Esempi:

10 NRF INIT 4 5
10 NRF INIT 4 5 76 250K HIGH 5

Note:

- Metti un **condensatore da 10–47 µF** vicino a VCC del modulo (indispensabile con PA/LNA).
- **CSN** deve essere **alto** quando usi altri dispositivi SPI (ILI9341, RC522, SD…).

NRF POWERDOWN

Sintassi:

NRF POWERDOWN

Descrizione:
Mette il modulo in **standby a bassissimo consumo**.

Esempio:

```
10 NRF POWERDOWN
20 WAIT 1000
30 NRF POWERUP
```

Note:

- Dopo POWERDOWN, chiama NRF POWERUP prima di trasmettere o ascoltare.

NRF POWERUP

Sintassi:

NRF POWERUP

Descrizione:
Riporta il modulo in stato **attivo** dal low-power.

Esempio:

```
10 NRF POWERUP
20 NRF START RX
```

Note:

- Dopo POWERUP servono alcuni ms prima che TX/RX siano pronti.

NRF READ

Sintassi:

NRF READ var$ [GOTO line]

Descrizione:

Se è presente un payload, lo legge in var$ (max 32 byte).
Se **non c'è nulla**:

- con GOTO line → salta a line.
- senza GOTO → imposta var$="" e prosegue.

Esempi:

```
10 NRF START RX
20 NRF READ M$ 100
30 PRINT "RX: ";M$
40 GOTO 20
100 WAIT 50 : GOTO 20
10 NRF START RX
20 NRF AVAILABLE A
30 IF A=1 THEN NRF READ M$
40 IF M$<>"" THEN PRINT "RX: ";M$
50 WAIT 50 : GOTO 20
```

Note:

- Con **DYNPL** attivo, legge esattamente la lunghezza ricevuta.
- Se ricevi dati binari, potresti trovarli non stampabili su seriale.

NRF SEND

Sintassi:

NRF SEND data$ [GOTO line]

Descrizione:

Invia data$ (max 32 byte). Se l'invio **fallisce** (niente ACK) ed è presente GOTO line, esegue il salto.

Esempi:

```
10 NRF SEND "HELLO" 200
20 PRINT "Inviato"
30 GOTO 10
200 PRINT "Errore invio"
10 LET MSG$ = "PING:" + STR$(TIMER)
20 NRF SEND MSG$
```

Note:

- Prima di inviare, assicurati di aver impostato NRF SET TXADDR e che il **peer** abbia aperto una RXADDR identica, con **AUTACK** coerente (di solito 1).
- Se stai ascoltando (START RX), la libreria passa temporaneamente in TX e poi torna in RX.

NRF SET RXADDR

Sintassi:

NRF SET RXADDR pipe "addr"

Descrizione:
Apre la **pipe di ricezione** 0..5 sull'indirizzo "addr" (3–5 caratteri ASCII o 0x... in esadecimale). Se almeno una pipe è aperta, parte l'ascolto.

Esempi:

10 NRF SET RXADDR 0 "NODE1"
10 NRF SET RXADDR 1 "0xAABBCCDDEE"

Note:

- Gli indirizzi nRF sono **little-endian** internamente; usare stringhe di **5 caratteri** è pratico e chiaro (es. "NODE1").
- Pipe 0 e 1 possono avere indirizzo completo; pipe 2..5 condividono i primi 4 byte con pipe1 nella libreria RF24 (gestito internamente).

NRF SET TXADDR

Sintassi:

NRF SET TXADDR "addr"

Descrizione:
Imposta l'indirizzo di **trasmissione** (3–5 char o 0x...).

Esempi:

10 NRF SET TXADDR "NODE1"
10 NRF SET TXADDR "0xE7E7E7E7E7"

Note:

- Per inviare con ACK, il ricevente deve avere una RXADDR **identica** attiva.
- Per topologie **stella**: fai trasmettere tutti verso l'**hub** (stesso TXADDR) e apri più **RXADDR** sull'hub per i nodi.

NRF START RX

Sintassi:

NRF START RX

Descrizione:
Entra in modalità **ascolto** (listening). Riceverai payload sulle pipe aperte.

Esempio:

```
10 NRF SET RXADDR 0 "NODE1"
20 NRF START RX
30 NRF READ M$ 100
40 IF M$<>"" THEN PRINT "RX: ";M$
50 GOTO 30
```

Note:

- Puoi aprire fino a **6 pipe** contemporanee (0..5).
- Se devi inviare mentre ascolti, NRF SEND sospende RX al bisogno.

NRF STOP RX

Sintassi:

```
NRF STOP RX
```

Descrizione:
Esce dalla modalità ascolto (torna pronto per TX).

Esempio:

```
10 NRF STOP RX
20 NRF SEND "HELLO"
```

Note:

- Non è obbligatorio prima di ogni NRF SEND (viene gestito), ma utile per script ordinati.

OLED CIRCLE

Sintassi:
OLED CIRCLE <x> <y> <raggio> [colore] [0|1]

Descrizione:
Disegna un cerchio non riempito centrato in <x>,<y> con raggio <raggio>.

Parametri opzionali:

- **[colore]**: WHITE (default) o BLACK
- **[0|1]**: aggiorna subito (1 o assente) oppure differisci (0)

Esempi pratici

Esempio 1 – Cerchio al centro dello schermo 128x64:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED CIRCLE 64 32 10 WHITE
```

Esempio 2 – Animazione cerchio che cresce:

```
10 OLED INIT 128 64 3C
20 FOR R = 2 TO 28
30 OLED CLEAR
40 OLED CIRCLE 64 32 R WHITE
50 WAIT 50
60 NEXT R
```

Note:

- Se vuoi cancellare un cerchio precedente, puoi ridisegnarlo in BLACK.
- Un cerchio pieno è disponibile con il comando OLED FILLCIRCLE.

OLED CLEAR

Sintassi:
OLED CLEAR

Descrizione:
Cancella completamente il contenuto visivo del display OLED.
Tutti i pixel vengono impostati a nero. **Non rimuove gli sprite** (che restano in memoria), ma azzera ciò che è mostrato sul display in quel momento.

Esempi pratici

Esempio 1 – Pulizia schermo:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
```

Esempio 2 – Pulizia ad ogni ciclo di animazione:

```
10 OLED INIT 128 64 3C
20 FOR R = 1 TO 16
30 OLED CLEAR
40 OLED CIRCLE 64 16 R
50 WAIT 100
60 NEXT R
```

Note:

- È consigliabile usarlo prima di disegnare nuovi elementi per evitare sovrapposizioni indesiderate.
- Non influisce sul contenuto degli sprite: se vuoi rivederli dopo un OLED CLEAR, devi ridisegnarli con OLED SPRITE DRAW (eventualmente bufferizzato) e poi OLED DRAW.

OLED DRAW

Sintassi:
OLED DRAW

Descrizione:
Aggiorna il pannello con tutto quello che è stato disegnato nel buffer ma non ancora mostrato.
Serve quando usi comandi con aggiornamento disabilitato (cioè quelli dove metti 0 come ultimo parametro): in quel caso il disegno resta nel buffer interno ma non viene ancora inviato al display fisico.
OLED DRAW manda tutto sullo schermo in un colpo solo (niente flicker).

Esempi pratici

Esempio pratico 1 – Linea bufferizzata:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED LINE 0 0 127 63 WHITE 0
40 OLED LINE 0 63 127 0 WHITE 0
50 OLED DRAW
```

Esempio pratico 2 – Testo, grafica e poi update unico:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED TEXT 10 10 1 "READY" WHITE 0
40 OLED RECT 0 0 128 64 WHITE 0
50 OLED DRAW
```

Note:

- Se usi sempre i comandi senza 0 finale (quindi aggiornamento immediato), non hai bisogno di OLED DRAW.

OLED DRAWRECT

Sintassi
OLED DRAWRECT x y w h

Descrizione
Aggiorna solo una porzione rettangolare del display OLED, copiando sullo schermo il contenuto già presente nel buffer grafico interno.
Il comando non disegna nulla nel buffer: rende visibili solo le modifiche già effettuate tramite comandi grafici o sprite.
È pensato per realizzare animazioni fluide e veloci usando la tecnica del dirty rectangle, evitando di aggiornare l'intero display.

Esempi pratici

Aggiorna l'intero display

```
10 OLED FILLRECT 0 0 128 64 WHITE 0
20 OLED DRAWRECT 0 0 128 64
```

Visualizza un rettangolo bianco a schermo.

Aggiorna solo una zona dello schermo

```
10 OLED FILLRECT 40 20 20 10 WHITE 0
20 OLED DRAWRECT 40 20 20 10
```

Aggiorna solo la porzione indicata del display.

Dirty rectangle con sprite che passa sopra una linea
In questo esempio:

- una linea orizzontale fa da sfondo
- uno sprite rettangolare si muove sopra la linea
- la linea non viene cancellata, perché la zona sporca viene ridisegnata correttamente nel buffer prima del DRAWRECT

```
10 OLED INIT 128 64 3C
20 REM ===== INIZIALIZZAZIONE BUFFER =====
30 OLED FILLRECT 0 0 128 64 BLACK 0
40 OLED LINE 0 32 127 32 WHITE 0
50 OLED DRAWRECT 0 0 128 64
60 REM ===== PARAMETRI QUADRATO =====
70 W = 10 : H = 10
80 X = 0 : Y = 28 : V = 2
90 REM ===== LOOP =====
100 OX = X
110 X = X + V
```

```
120 IF X <= 0 THEN V = -V
130 IF X >= (128-W) THEN V = -V
140 REM calcolo dirty rectangle
150 XD = OX : IF X < OX THEN XD = X
160 WD = X - OX : IF WD < 0 THEN WD = -WD
170 WD = WD + W
180 REM ripristina sfondo SOLO nella zona sporca
190 OLED FILLRECT XD Y WD H BLACK 0
200 REM ridisegna il pezzo di linea sotto (con coordinate GIUSTE)
210 X2 = XD + WD - 1
220 OLED LINE XD 32 X2 32 WHITE 0
230 REM disegna quadrato nella nuova posizione (buffer)
240 OLED FILLRECT X Y W H WHITE 0
250 REM aggiorna SOLO la zona sporca
260 OLED DRAWRECT XD Y WD H
270 WAIT 20
280 GOTO 100
```

Durante il movimento:

- il rettangolo passa sopra la linea
- la linea viene correttamente ridisegnata
- nessuna scia e nessun flicker

Note

- OLED DRAWRECT non cancella né disegna oggetti: aggiorna solo il display.
- È responsabilità del programma ripristinare il contenuto corretto del buffer prima del refresh.
- In presenza di sprite, il flusso corretto è:
    1. pulire la zona sporca nel buffer
    2. ridisegnare lo sfondo nella zona
    3. ridisegnare lo sprite
    4. chiamare OLED DRAWRECT
- Usare OLED DRAWRECT su piccole aree migliora notevolmente le prestazioni rispetto a OLED DRAW.

OLED FILLCIRCLE

Sintassi:
OLED FILLCIRCLE <x> <y> <raggio> [colore] [0|1]

Descrizione:
Disegna un cerchio pieno (riempito) centrato in <x>,<y> con raggio <raggio>.
Questo comando colora tutti i pixel all'interno del cerchio.

Parametri opzionali:

- **[colore]**: WHITE (default) o BLACK
  - WHITE riempie il cerchio in bianco
  - BLACK riempie in nero e quindi può essere usato per cancellare una zona circolare
- **[0|1]**: aggiornare subito lo schermo o no
  - 1 (o assente): disegna e aggiorna immediatamente il display
  - 0: disegna solo nel buffer; per visualizzare bisogna poi chiamare OLED DRAW

Esempi pratici

Esempio 1 – Pallino pieno bianco al centro dello schermo:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLCIRCLE 64 32 8 WHITE
```

Risultato: cerchio pieno bianco con raggio 8 al centro del display.

Esempio 2 – "Bolla nera" senza aggiornamento immediato:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLRECT 0 0 128 64 WHITE 0
40 OLED FILLCIRCLE 30 30 10 BLACK 0
50 OLED DRAW
```

Esempio 3 – Effetto "target" (cerchi pieni concentrici)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLCIRCLE 64 32 12 WHITE 0
40 OLED FILLCIRCLE 64 32 8 BLACK 0
50 OLED FILLCIRCLE 64 32 4 WHITE 0
60 OLED DRAW
```

Note:

- È diverso da OLED CIRCLE, che disegna solo il bordo del cerchio.
- Con BLACK puoi “bucare” o cancellare aree tonde dentro grafica già disegnata.
- Con aggiornamento 0 puoi preparare grafica complessa e poi fare OLED DRAW per mostrarla in un colpo solo (senza flicker).

OLED FILLRECT

Sintassi:
OLED FILLRECT <x> <y> <larghezza> <altezza> [colore] [0|1]

Descrizione:
Disegna un rettangolo pieno alle coordinate date.
Serve per riempire aree, fare pannelli, o cancellare zone.

Parametri opzionali:

- **[colore]**: WHITE o BLACK (default: WHITE)
- **[0|1]**: aggiornare subito (1 o assente) oppure dopo (0)

Esempi pratici

Esempio 1 – Barra bianca:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED FILLRECT 0 56 128 8 WHITE
```

Esempio 2 – Cancella una zona ma non aggiornare ancora:

```
10 OLED FILLRECT 0 56 128 8 BLACK 0
20 OLED DRAW
```

Note:

- Disegnando BLACK copri e cancelli quello che c'era sotto.
- Molto utile per status bar nere, riquadri, box di testo, ecc.
- Usato con update=0 è perfetto per costruire una schermata intera invisibile e poi renderla visibile tutta insieme con OLED DRAW.

OLED INIT (DISPLAY SSD1306)

Sintassi:
OLED INIT <larghezza> <altezza> <indirizzo_hex> [sda scl]

Descrizione:
Inizializza il display OLED SSD1306 via I2C.
Dopo questo comando il display viene preparato, cancellato e reso pronto all'uso.

Parametri:

- <larghezza>: larghezza in pixel (tipico: 128)
- <altezza>: altezza in pixel (tipico: 64)
- <indirizzo_hex>: indirizzo I2C del display (di solito 3C)
  - puoi scriverlo come 3C o 0x3C
- [sda scl] (opzionale): pin SDA e SCL per l'I2C
  - default ESP32: SDA=21, SCL=22

Questo comando deve essere chiamato prima di usare qualsiasi altro comando OLED.

Esempi pratici

Esempio 1 – Inizializzazione standard 128x64:

10 OLED INIT 128 64 3C

Esempio 2 – Inizializzazione forzando i pin I2C:

10 OLED INIT 128 64 3C 21 22

Note:

- Dopo l'inizializzazione, il firmware abilita l'OLED, quindi tutti gli altri comandi diventano attivi.
- L'indirizzo più comune per SSD1306 è 0x3C.
- Il display viene pulito automaticamente.

OLED INVERT ON / OFF

Sintassi:
OLED INVERT ON
OLED INVERT OFF

Descrizione:
Attiva o disattiva la modalità invertita del display OLED:

- ON: sfondo bianco, testo nero
- OFF: sfondo nero, testo bianco (standard)

Esempi pratici

```
10 OLED INIT 128 64 3C
20 OLED INVERT ON
30 OLED TEXT 10 10 1 "INVERTITO"
40 DELAY 1000
50 OLED INVERT OFF
```

Note:

- Non cancella il contenuto del display.
- È utile per effetti temporanei o focus visivo su un’area.

OLED LINE

Sintassi:
OLED LINE <x1> <y1> <x2> <y2> [colore] [0|1]

Descrizione:
Disegna una linea dal punto <x1>,<y1> al punto <x2>,<y2>.

Parametri opzionali:

- **[colore]**: WHITE (default) o BLACK
- **[0|1]**: aggiornare subito (1/default) o rimandare (0)

Esempi pratici

Esempio 1 – Diagonale sullo schermo:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED LINE 0 0 127 63
```

Esempio 2 – Linea orizzontale bianca bufferizzata:

```
10 OLED INIT 128 64 3C
20 OLED LINE 0 32 127 32 WHITE 0
30 OLED DRAW
```

Note:

- Con BLACK puoi cancellare una linea precedente bianca nella stessa posizione.
- Puoi disegnare più linee con aggiornamento 0 e poi chiamare OLED DRAW.

OLED PIXEL

Sintassi:
OLED PIXEL <x> <y> [colore] [0|1]

Descrizione:
Disegna un singolo pixel nelle coordinate <x>,<y>.

Parametri opzionali:

- **[colore]**: WHITE o BLACK (default: WHITE)
- **[0|1]**: aggiorna subito lo schermo?
  - 1 (o assente): aggiorna subito
  - 0: non aggiorna ancora (serve poi OLED DRAW)

Esempi pratici

Esempio 1 – Pixel bianco con aggiornamento immediato:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED PIXEL 10 10
```

Esempio 2 – Pixel bianco senza aggiornare subito:

```
10 OLED INIT 128 64 3C
20 OLED PIXEL 20 20 WHITE 0
30 OLED DRAW
```

Note:

- Con BLACK puoi “spegnere” un pixel.
- Se <x> o <y> sono fuori schermo, il comando non disegna niente.

OLED RECT

Sintassi:
OLED RECT <x> <y> <larghezza> <altezza> [colore] [0|1]

Descrizione:
Disegna un rettangolo vuoto (solo bordo) con angolo superiore sinistro in <x>,<y>, larghezza <larghezza> e altezza <altezza>.

Parametri opzionali:

- **[colore]**: WHITE (default) o BLACK
- **[0|1]**: aggiornamento immediato o ritardato

Esempi pratici

Esempio 1 – Bordo dello schermo:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED RECT 0 0 128 64
```

Esempio 2 – Rettangolo nero (per cancellare un bordo precedente):

```
10 OLED RECT 0 0 128 64 BLACK 1
```

Note:

- OLED RECT disegna solo il contorno. Per un rettangolo riempito usa OLED FILLRECT.
- Puoi combinare più ... 0 e poi OLED DRAW.

OLED SPRITE CLEAR

Sintassi:
OLED SPRITE CLEAR

Descrizione:
Rimuove tutti gli sprite attivi dalla memoria. Dopo questo comando, nessuno sprite verrà disegnato fino alla loro ricreazione con OLED SPRITE NEW.

Esempi pratici

Esempio 1 – Pulisce tutti gli sprite

```
10 OLED SPRITE CLEAR
```

Note

- Il comando rimuove gli sprite, ma **non aggiorna automaticamente il display**.
- Se vuoi anche pulire lo schermo, usa OLED CLEAR e/o ridisegna la scena e poi fai OLED DRAW.

OLED SPRITE COLLISION

Sintassi

OLED SPRITE COLLISION id1 TO id2 GOTO line
OLED SPRITE COLLISION id1 TO ANY GOTO line
OLED SPRITE COLLISION id1 TO id2 SET variabile
OLED SPRITE COLLISION id1 TO ANY SET variabile

Descrizione

Verifica la collisione **rettangolare** (AABB) tra sprite disegnati sull'OLED.
In caso di collisione:

- con **GOTO** salta alla riga indicata;
- con **SET variabile** assegna `1` (collisione) o `0` (nessuna collisione) a una variabile (numerica o stringa `$`).

Esempi pratici (copia-incolla)

Esempio – GOTO (rettangolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
40 OLED SPRITE NEW 2 RECT 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE"
210 END
```

Esempio – SET variabile (rettangolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
40 OLED SPRITE NEW 2 RECT 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO 2 SET C
67 IF C THEN PRINT "Colpito!" ELSE PRINT "Libero"
```

```
70 GOTO 53
```

Esempio – ANY con SET (rettangolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 RECT 10 30 8 8
31 OLED SPRITE NEW 2 RECT 80 30 8 8
32 OLED SPRITE NEW 3 RECT 100 10 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISION 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 ha urtato qualcuno!"
70 GOTO 53
```

Note

- Gli **ID** devono riferirsi a sprite **attivi** (OLED SPRITE NEW).
- Se la variabile termina con $, riceve "1"/"0"; altrimenti 1.0/0.0.
- AABB è molto veloce; su 128×64 l’impatto è minimo.
- Per prestazioni stabili con molti sprite, limita i test per ciclo.

OLED SPRITE COLLISIONC

Sintassi

OLED SPRITE COLLISIONC id1 TO id2 GOTO line
OLED SPRITE COLLISIONC id1 TO ANY GOTO line
OLED SPRITE COLLISIONC id1 TO id2 SET variabile
OLED SPRITE COLLISIONC id1 TO ANY SET variabile

Descrizione

Verifica la collisione **circolare** usando i cerchi **inscritti** nei bounding box degli sprite: centro = centro bbox, raggio = min(larghezza, altezza)/2.
Utile per sprite rotondi o per una rilevazione più “morbida”.

Esempi pratici (copia-incolla)

Esempio – GOTO (circolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8 8
40 OLED SPRITE NEW 2 CIRCLE 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO 2 GOTO 200
70 GOTO 53
200 PRINT "COLLISIONE CIRCOLARE"
210 END
```

Esempio – SET variabile (circolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8 8
40 OLED SPRITE NEW 2 CIRCLE 80 30 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO 2 SET CIRC
67 IF CIRC THEN PRINT "Contatto (cerchi)!" ELSE PRINT "Nessun contatto"
70 GOTO 53
```

Esempio – ANY con SET (circolare)

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED SPRITE NEW 1 CIRCLE 10 30 8 8
31 OLED SPRITE NEW 2 CIRCLE 80 30 8 8
32 OLED SPRITE NEW 3 CIRCLE 100 10 8 8
50 OLED SPRITE DRAW
51 X = 10
52 Y = 30
53 X = X+1
60 OLED SPRITE MOVE 1 X Y
65 OLED SPRITE DRAW
66 OLED SPRITE COLLISIONC 1 TO ANY SET HIT
67 IF HIT THEN PRINT "1 ha toccato almeno uno (circolare)"
70 GOTO 53
```

Note

- Sprite **attivi** e bbox valido richiesti.
- Variabili stringa (VAR$) → "1"/"0"; numeriche (VAR) → 1/0.
- COLLISIONC è leggermente più costosa di COLLISION, ma su OLED l'impatto è trascurabile.
- Mantieni l'ordine dei rami nel parser: prima TO ANY SET, poi SET, infine GOTO.

OLED SPRITE DELETE

Sintassi:

OLED SPRITE DELETE <id>

Descrizione:

Disattiva e rimuove lo sprite con identificativo <id>.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 10
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE DELETE 0
70 OLED SPRITE DRAW
```

Note:

- Lo sprite viene eliminato e non sarà più disegnato finché non verrà ricreato.
- L'id deve essere valido (da 0 a MAX_SPRITES-1).

OLED SPRITE DRAW

Sintassi:

OLED SPRITE DRAW
OLED SPRITE DRAW id
OLED SPRITE DRAW ALL 0
OLED SPRITE DRAW id 0

Descrizione:
Disegna gli sprite OLED nel framebuffer interno oppure direttamente sul display, in base alla modalità di aggiornamento.

Il comando può:

- disegnare tutti gli sprite attivi e visibili
- oppure un solo sprite specifico
- con aggiornamento immediato (default)
- oppure senza aggiornare subito il display (modalità bufferizzata)

Deve essere chiamato dopo qualsiasi modifica agli sprite
(creazione, spostamento, cambio contenuto, bitmap, testo, ecc.).

Modalità di aggiornamento:

Sintassi	Comportamento
OLED SPRITE DRAW	Disegna tutti gli sprite e aggiorna subito il display
OLED SPRITE DRAW id	Disegna lo sprite indicato e aggiorna subito il display
OLED SPRITE DRAW ALL 0	Disegna tutti gli sprite nel buffer, senza aggiornare il display
OLED SPRITE DRAW id 0	Disegna solo lo sprite indicato nel buffer, senza aggiornare il display

Per rendere visibile il contenuto bufferizzato è necessario chiamare:
OLED DRAW
oppure, per aggiornare solo una zona:
OLED DRAWRECT x y w h

Esempi pratici

Esempio 1 – Disegno immediato (comportamento classico)

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 40
30 OLED SPRITE SHOW 0
```

```
40 OLED SPRITE DRAW
```

Esempio 2 – Disegno bufferizzato (no flicker)

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 40
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW ALL 0
50 OLED DRAW
```

Esempio 3 – Animazione fluida senza flicker

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 RECT 0 28 10 10
30 OLED SPRITE SHOW 0
40 X = 0
50 V = 2
60 X = X + V
70 IF X <= 0 THEN V = -V
80 IF X >= 118 THEN V = -V
90 OLED SPRITE MOVE 0 X 28
100 OLED FILLRECT 0 0 128 64 BLACK 0
110 OLED SPRITE DRAW ALL 0
120 OLED DRAW
130 WAIT 20
140 GOTO 60
```

Esempio 4 – Uso avanzato con OLED DRAWRECT (dirty-rect)

```
10 OLED SPRITE DRAW ALL 0
20 OLED DRAWRECT 40 20 20 20
```

Note:

- Se usato senza 0, OLED SPRITE DRAW aggiorna subito il display (può causare flicker se chiamato più volte nello stesso frame).
- Se usato con 0, il disegno avviene solo nel buffer; per visualizzarlo serve OLED DRAW o OLED DRAWRECT.
- Per animazioni fluide: usare sempre OLED SPRITE DRAW ALL 0 e poi una sola OLED DRAW (o OLED DRAWRECT) per frame.
- L’ordine di disegno degli sprite dipende dall’ID: ID più alto = sopra.

OLED SPRITE HIDE

Sintassi:

OLED SPRITE HIDE <id>

Descrizione:

Nasconde temporaneamente lo sprite con id senza cancellarlo.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 RECT 10 10 40 10
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE HIDE 0
70 OLED SPRITE DRAW
```

Note:

- Lo sprite esiste ancora ma non viene disegnato.

OLED SPRITE MOVE

Sintassi:

OLED SPRITE MOVE <id> <x> <y>

Descrizione:

Sposta lo sprite identificato da <id> alla posizione <x>, <y>.

Esempi pratici

```
10 OLED INIT 128 64 3C
20 OLED SPRITE NEW 0 FRAME 0 10 40 10
30 OLED SPRITE SHOW 0
40 FOR X = 0 TO 80 STEP 10
50 OLED SPRITE MOVE 0 X 10
60 OLED SPRITE DRAW
70 DELAY 100
80 NEXT X
```

Note:

- Per rendere effettivo lo spostamento, chiamare OLED SPRITE DRAW.

OLED SPRITE NEW

Sintassi generale

OLED SPRITE NEW id tipo x y parametri

- **id** → identificatore univoco dello sprite
- **tipo** → RECT, FRAME, CIRCLE, LINE, TEXT, CHAR, NUM, BITMAP
- **x y** → posizione dello sprite sul display OLED
- **parametri** → variano in base al tipo

Gli sprite vengono visualizzati tramite:

OLED SPRITE DRAW

(oppure singolarmente con OLED SPRITE SHOW id)

---

1. RETTANGOLO PIENO (RECT)

Sintassi

OLED SPRITE NEW id RECT x y larghezza altezza

Descrizione
Disegna un rettangolo pieno.

Esempio

20 OLED SPRITE NEW 0 RECT 10 10 30 8

→ Rettangolo pieno 30×8 alla posizione (10,10).

---

2. CORNICE (FRAME)

Sintassi

OLED SPRITE NEW id FRAME x y larghezza altezza

Descrizione
Disegna il contorno di un rettangolo.

Esempio

30 OLED SPRITE NEW 1 FRAME 50 8 30 8

→ Cornice 30×8 alla posizione (50,8).

---

3. CERCHIO (CIRCLE)

Sintassi

OLED SPRITE NEW id CIRCLE x y raggio

Descrizione
Disegna un **cerchio pieno** con centro in (x,y).

Esempio

40 OLED SPRITE NEW 2 CIRCLE 90 16 6

→ Cerchio pieno di raggio 6 centrato in (90,16).

Nota

- A differenza di OLED CIRCLE (non sprite), lo sprite CIRCLE è sempre pieno.

---

4. LINEA (LINE)

Sintassi

OLED SPRITE NEW id LINE x y dx dy

Descrizione
Disegna una linea dal punto (x,y) al punto (x+dx, y+dy).

Esempio

50 OLED SPRITE NEW 3 LINE 0 0 30 16

→ Linea da (0,0) a (30,16).

---

5. TESTO (TEXT)

Sintassi

OLED SPRITE NEW id TEXT x y size "testo"

Descrizione
Mostra una stringa di testo con font scalabile.

Esempio

60 OLED SPRITE NEW 4 TEXT 0 24 1 "HELLO"

→ Testo “HELLO”, grandezza 1, a (0,24).

---

6. CARATTERE SINGOLO (CHAR)

Sintassi

OLED SPRITE NEW id CHAR x y "carattere" size

Descrizione
Mostra un singolo carattere.

Esempio

70 OLED SPRITE NEW 5 CHAR 90 24 "A" 2

→ Mostra “A” grandezza 2 a (90,24).

---

7. NUMERO (NUM)

Sintassi

OLED SPRITE NEW id NUM x y numero size

Descrizione
Mostra un numero (intero o reale).

Esempio

80 OLED SPRITE NEW 6 NUM 110 0 2024 1

→ Mostra “2024”, grandezza 1, a (110,0).

---

8. BITMAP (BITMAP)

Sintassi

OLED SPRITE NEW id BITMAP x y larghezza altezza pattern scala

Descrizione
Disegna una piccola immagine monocromatica basata su una stringa di pattern (0/1 o cifre/pixel).

Pattern di esempio

P$ = "0110" + "1001" + "1001" + "0110"

(4×4 pixel)

Esempio

100 OLED SPRITE NEW 7 BITMAP 40 40 4 4 P$ 4

→ Bitmap 4×4 scalata ×4 (16×16 pixel) a (40,40).

Esempio completo di programma

```
10 OLED INIT 128 64 3C

20 OLED SPRITE NEW 0 RECT 10 10 30 8
30 OLED SPRITE NEW 1 FRAME 50 8 30 8
40 OLED SPRITE NEW 2 CIRCLE 90 16 6
50 OLED SPRITE NEW 3 LINE 0 0 30 16
60 OLED SPRITE NEW 4 TEXT 0 24 1 "HELLO"
70 OLED SPRITE NEW 5 CHAR 90 24 "A" 2
80 OLED SPRITE NEW 6 NUM 110 0 2024 1

90 P$ = "0110" + "1001" + "1001" + "0110"
100 OLED SPRITE NEW 7 BITMAP 40 40 4 4 P$ 4

110 FOR I = 0 TO 7
120   OLED SPRITE SHOW I
130 NEXT I

140 OLED SPRITE DRAW
150 END
```

NOTE

- Gli sprite OLED sono monocromatici.
- OLED SPRITE SHOW id abilita singoli sprite.
- OLED SPRITE DRAW renderizza tutti gli sprite attivi e visibili.
- Sono supportate variabili numeriche e stringhe.
- I bitmap usano pattern concatenati senza spazi.

OLED SPRITE SETBITMAP

Sintassi:

OLED SPRITE SETBITMAP id pattern$

Descrizione:
Cambia il pattern grafico di uno sprite BITMAP già creato.
Serve per animazioni (camminata, movimento, sprite del serpente, oggetti che cambiano forma…).

Esempi pratici

Animazione semplice (2 fotogrammi)

```
10 P1$ = "01110" + "10001" + "11111" + "10001" + "01110"
20 P2$ = "01110" + "11011" + "11111" + "11011" + "01110"
30 OLED SPRITE NEW 0 BITMAP 40 40 5 5 P1$ 4
40 FOR I = 1 TO 10
50   OLED SPRITE SETBITMAP 0 P1$
60   OLED SPRITE DRAW 0
70   DELAY 100
80   OLED SPRITE SETBITMAP 0 P2$
90  OLED SPRITE DRAW 0
100  DELAY 100
110 NEXT I
```

Note

- Lo sprite mantiene: posizione, scala, dimensioni, visibilità.
- Cambia *solo* il pattern.
- Il nuovo pattern deve avere la stessa dimensione W×H dello sprite originario.
- Per vedere il cambio serve un OLED SPRITE DRAW id.

OLED SPRITE SETCHAR

Sintassi:

OLED SPRITE SETCHAR <id> "X"

Descrizione:

Modifica il carattere mostrato da uno sprite di tipo CHAR.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 2 CHAR 100 0 "A" 1
30 OLED SPRITE SHOW 2
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETCHAR 2 "B"
70 OLED SPRITE DRAW
```

OLED SPRITE SETNUM

Sintassi:

OLED SPRITE SETNUM <id> <numero>

Descrizione:

Aggiorna il numero visualizzato da uno sprite di tipo NUM.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 1 NUM 60 0 123 1
30 OLED SPRITE SHOW 1
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETNUM 1 2024
70 OLED SPRITE DRAW
```

OLED SPRITE SETTEXT

Sintassi:

OLED SPRITE SETTEXT <id> "nuovo testo"

Descrizione:

Modifica il contenuto testuale dello sprite TEXT.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 TEXT 0 0 1 "HELLO"
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETTEXT 0 "WORLD"
70 OLED SPRITE DRAW
```

OLED SPRITE SHOW

Sintassi:

OLED SPRITE SHOW <id>

Descrizione:

Rende visibile uno sprite precedentemente nascosto o creato.

Esempi pratici

```
10 OLED INIT 128 32 3C
20 OLED SPRITE NEW 0 CIRCLE 20 16 6
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
```

OLED TEXT

Sintassi:
OLED TEXT <x> <y> <dimensione> <testo/expr> [colore] [0|1]

Descrizione:
Scrive testo sul display alla posizione <x>,<y> con dimensione carattere <dimensione>.

Il parametro <testo/expr> può essere:

- una stringa tra virgolette
  OLED TEXT 0 0 1 "CIAO"
- una variabile stringa
  OLED TEXT 0 16 1 A$
- una concatenazione con +
  OLED TEXT 0 0 1 "TEMP=" + T + "C"
- una funzione stringa del BASIC
  OLED TEXT 0 0 1 LEFT$(A$,3)
- un numero o un'espressione numerica (viene convertita in stringa automaticamente)

Parametri opzionali finali:

- [colore]: WHITE o BLACK (default: WHITE)
- [0|1]: aggiorna 1 (o assente), oppure bufferizza 0

Esempi pratici

Esempio 1 – Titolo grande:

```
10 OLED INIT 128 64 3C
20 OLED CLEAR
30 OLED TEXT 20 8 2 "HELLO" WHITE
```

Esempio 2 – Testo costruito con variabili e aggiornamento ritardato:

```
10 T = 22
20 MSG$ = "READY"
30 OLED TEXT 0 0 1 "TEMP=" + T + "C" WHITE 0
40 OLED TEXT 0 16 1 MSG$ WHITE 0
50 OLED DRAW
```

Esempio 3 – Scrivi e cancella:

```
10 OLED TEXT 0 32 1 "LOADING..." WHITE
20 WAIT 1000
30 OLED FILLRECT 0 32 80 10 BLACK
```

40 OLED TEXT 0 32 1 "DONE" WHITE

Note:

- dimensione influisce su larghezza e altezza dei caratteri (1 = normale, 2 = ingrandito).
- Con update=0 puoi scrivere più righe di testo sul buffer e mostrarle tutte insieme con OLED DRAW.

ON x GOTO ...

(o ON x GOSUB ...)

Sintassi:

ON espressione GOTO riga1, riga2, riga3, ...
ON espressione GOSUB riga1, riga2, riga3, ...

Descrizione:

ON x GOTO (o ON x GOSUB) esegue un **salto condizionato** alla riga corrispondente alla posizione x.
Funziona come un **menu numerico** o uno **switch-case**, selezionando la riga da eseguire in base al valore di x.

- Se x = 1, salta alla **prima riga** elencata
- Se x = 2, salta alla **seconda riga**, e così via
- Se x è fuori intervallo, **non succede nulla**

Esempi pratici

Esempio 1 – GOTO condizionato

```
10 INPUT X
20 ON X GOTO 100, 200, 300
30 PRINT "NESSUNA SCELTA VALIDA"
40 END
100 PRINT "SCELTO 1": END
200 PRINT "SCELTO 2": END
300 PRINT "SCELTO 3": END
RUN
```

Output atteso (se input = 2):

```
SCELTO 2
```

Esempio 2 – GOSUB condizionato

→ Richiama una subroutine diversa in base al valore:

```
10 INPUT N
20 ON N GOSUB 100, 200
30 PRINT "RITORNO AL MAIN"
40 END
100 PRINT "FUNZIONE A": RETURN
200 PRINT "FUNZIONE B": RETURN
RUN
```

Output atteso (se input = 1):

```
FUNZIONE A
```

RITORNO AL MAIN

Esempio 3 – Valore fuori intervallo

→ Se x è zero o maggiore del numero di opzioni, il salto **non avviene**:

```
10 X = 0
20 ON X GOTO 100, 200
30 PRINT "X NON VALIDO"
RUN
```

Output atteso:

```
X NON VALIDO
```

Esempio 4 – Uso con variabile numerica da calcolo

```
10 A = INT(RND(1) * 3) + 1
20 ON A GOTO 100, 200, 300
100 PRINT "PRIMO": END
200 PRINT "SECONDO": END
300 PRINT "TERZO": END
RUN
```

Output atteso:

Una delle tre righe viene eseguita casualmente.

Nota:

- ON ... GOTO può avere da 1 a 255 salti indicati
- Se usi ON ... GOSUB, ricorda sempre il RETURN alla fine della subroutine

OPEN

Sintassi

OPEN [SPIFFS | SD] "<filename>" <mode>

Descrizione

Apre un file di testo sul filesystem scelto:

- SPIFFS → memoria interna dell'ESP32
- SD → scheda SD
- Se non specifichi nulla → default **SPIFFS**

<mode> può essere:

Mod	Significato
R	Lettura (read)
W	Scrittura (write, sovrascrive tutto)
A	Append (scrive a fine file)

Se un altro file era già aperto, viene automaticamente chiuso.

Esempi pratici

Esempio 1 – Scrivere un file

```
10 OPEN SPIFFS "log.txt" W
20 FPRINT "CIAO"
30 FPRINT "VALORE=42"
40 FCLOSE
```

Esempio 2 – Append su SD

```
10 OPEN SD "storia.txt" A
20 FPRINT "Nuova riga"
30 FCLOSE
```

Esempio 3 – Default SPIFFS

```
10 OPEN "note.txt" W
20 FPRINT "Hello!"
30 FCLOSE
```

Note

- Un solo file può essere aperto alla volta.
- Modalità **A** permette anche la lettura prima dell'append.
- Il filename è sempre una STRINGA tra virgolette.
- Se il filesystem non è inizializzato (es. SD non montata), viene generato un errore.

PEEK

Sintassi:

PEEK(indirizzo)

Descrizione:

La funzione PEEK legge il **valore numerico (byte)** contenuto in una determinata **locazione di memoria**.
È il complemento di POKE, usato per ottenere ciò che è stato precedentemente scritto o per leggere aree di memoria mappata.

Il valore restituito è compreso tra 0 e 255.

Esempi pratici

Esempio 1 – Lettura di un valore dopo POKE

```
10 POKE 1000, 88
20 PRINT "VALORE INDIRIZZO 1000: "; PEEK(1000)
RUN
```

Output atteso:

```
VALORE INDIRIZZO 1000: 88
```

Esempio 2 – Lettura diretta

→ Se un valore era stato precedentemente scritto:

```
10 PRINT PEEK(2000)
RUN
```

Output atteso:

Valore attualmente memorizzato all'indirizzo 2000.

Esempio 3 – Ciclo di lettura

```
10 FOR I = 0 TO 4
20 POKE 3000 + I, I * 2
30 NEXT I
40 FOR I = 0 TO 4
50 PRINT "PEEK("; 3000 + I; ") = "; PEEK(3000 + I)
60 NEXT I
RUN
```

Output atteso:

```
PEEK(3000) = 0
PEEK(3001) = 2
PEEK(3002) = 4
PEEK(3003) = 6
PEEK(3004) = 8
```

Esempio 4 – Lettura di un’area vuota

```
10 PRINT "VALORE: "; PEEK(5000)
RUN
```

Output atteso:
Il valore dipende dallo stato della memoria (potrebbe essere `0` o altro).

Nota:

- È sempre bene accertarsi che l’indirizzo letto sia valido nel contesto del firmware
- Per modificare un valore in memoria, usa `POKE`

PINMODE

Sintassi:

PINMODE pin INPUT|OUTPUT NOPULL|PULLUP|PULLDOWN DEBOUNCE ms

Descrizione:
Configura un GPIO dell'ESP32:

- **modo**: INPUT (ingresso) oppure OUTPUT (uscita)
- **resistenza**: NOPULL, PULLUP, PULLDOWN
- **DEBOUNCE [ms]** *(opzionale, solo per INPUT)*: abilita l'anti-rimbalzo **one-shot** per DREAD(pin).
    - Con **DEBOUNCE**, DREAD(pin) genera **un solo evento per pressione** e non ripete finché non rilasci.
    - Senza **DEBOUNCE**, DREAD(pin) restituisce il **livello raw** e può ripetere mentre tieni premuto.
    - ms è il tempo di stabilizzazione (default **50 ms** se omesso).

Con **PULLUP**: premuto = **LOW (0)**, rilasciato = **HIGH (1)**.
Con **PULLDOWN**: premuto = **HIGH (1)**, rilasciato = **LOW (0)**.
DLEVEL(pin) restituisce **sempre** il livello raw, ignorando il debounce.

Esempi pratici

Esempio 1 – OUTPUT senza resistenze
Accende un LED su GPIO2.

```
10 PINMODE 2 OUTPUT NOPULL
20 DWRITE 2 1
```

Esempio 2 – INPUT con PULLUP e DEBOUNCE (one-shot)
Conta una pressione alla volta sul GPIO5.

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DREAD(12) = 0 THEN PRINT “PULSANTE PREMUTO”
30 GOTO 20
```

Esempio 3 – INPUT con PULLUP senza DEBOUNCE (raw)
Ripete finché tieni premuto.

```
10 PINMODE 12 INPUT PULLUP
20 IF DREAD(12) = 0 THEN PRINT "PULSANTE PREMUTO”
30 GOTO 20
```

Esempio 4 – Stato istantaneo con DLEVEL (ignora debounce)
Mostra “tenuto premuto” in tempo reale.

```
10 PINMODE 12 INPUT PULLUP DEBOUNCE 60
20 IF DLEVEL(12) = 0 THEN PRINT "HOLD" ELSE PRINT "UP"
30 DELAY 500
40 GOTO 20
```

Esempio 5 – PULLDOWN + DEBOUNCE (one-shot su 1)
Con sensore attivo-alto.

```
10 PINMODE 12 INPUT PULLDOWN DEBOUNCE 50
20 IF DREAD(12) = 1 THEN PRINT "EVENTO"
30 GOTO 20
```

Note

- Usa sempre PINMODE **prima** di DREAD, DLEVEL, DWRITE.
- DEBOUNCE vale **solo per INPUT** e modifica il comportamento di **DREAD(pin)** (one-shot); **DLEVEL(pin)** resta raw.
- Il valore ms (se specificato) imposta la finestra anti-rimbalzo; se omesso, **50 ms**.
- Su alcune schede **PULLDOWN** hardware può non essere disponibile su tutti i pin: in tal caso usare resistenze esterne.

POKE

Sintassi

POKE indirizzo, valore

Descrizione

Il comando **POKE** scrive un valore numerico (**1 byte**) in una **memoria virtuale BASIC interna**, dedicata all'uso di PEEK e POKE.

Questa memoria **non corrisponde alla RAM reale dell'ESP32**, ma a un'area protetta e controllata dal firmware BASIC, progettata per:

- simulare il comportamento dei BASIC classici (C64, ZX, ecc.)
- creare buffer temporanei
- memorizzare dati binari o strutture semplici
- evitare crash o accessi non validi alla memoria del microcontrollore

Parametri

- **indirizzo**
  Numero intero che indica la posizione di memoria da modificare.
  Deve essere compreso tra **0 e 511**.
- **valore**
  Numero intero compreso tra **0 e 255**.
  Viene scritto come **byte** nella locazione indicata.

Se l'indirizzo è fuori dal range consentito, il comando genera un errore di runtime.

Memoria disponibile

La memoria virtuale BASIC per POKE e PEEK è di:

512 byte (indirizzi da 0 a 511)

Ogni indirizzo contiene **un singolo byte**.

Esempi pratici

Esempio 1 – Scrittura di un valore

```
10 POKE 100, 42
RUN
```

Scrive il valore **42** nella posizione di memoria **100**.
Nessun output a video.

Esempio 2 – Scrittura seguita da lettura con PEEK

```
10 POKE 200, 77
20 PRINT "VALORE IN MEMORIA: "; PEEK(200)
RUN
```

Output atteso

```
VALORE IN MEMORIA: 77
```

Esempio 3 – Uso in un ciclo

Riempie una serie di locazioni contigue:

```
10 FOR I = 0 TO 9
20 POKE I, I * 2
30 NEXT I
RUN
```

Effetto:

```
INDIRIZZO 0  = 0
INDIRIZZO 1  = 2
INDIRIZZO 2  = 4
...
INDIRIZZO 9  = 18
```

Esempio 4 – Simulazione di buffer

```
10 FOR I = 0 TO 4
20 POKE 100 + I, I * 10
30 NEXT I
40 FOR I = 0 TO 4
50 PRINT PEEK(100 + I)
60 NEXT I
RUN
```

Output atteso

```
0
10
20
30
40
```

Errori comuni

Indirizzo fuori range

POKE 600, 1

Errore:

POKE: address out of range (0-511)

Nota importante

- POKE opera **solo sulla memoria virtuale BASIC**
- Non è possibile accedere a RAM, registri o periferiche dell’ESP32
- Questo garantisce stabilità e sicurezza del firmware
- Per leggere i valori scritti, usare il comando complementare **PEEK**

PRINT

(o ?)

Sintassi:

PRINT espressione
PRINT variabile [,|;] ...

Descrizione:
PRINT visualizza a schermo (sul terminale seriale) **testo, numeri, risultati di espressioni o variabili**.
È uno dei comandi fondamentali per:

- **mostrare messaggi**
- **visualizzare risultati**
- **fare debug**

Può stampare stringhe ("Testo"), numeri (123, A), combinazioni ("X = "; X) e supporta:

- ; → stampa sulla stessa riga senza spazio
- , → allinea alla colonna successiva

Esempi pratici

Esempio 1 – Stampa semplice di una stringa

```
10 PRINT "CIAO MONDO"
RUN
```

Output atteso:

```
CIAO MONDO
```

Esempio 2 – Stampa di un numero e un testo

```
10 A = 5
20 PRINT "VALORE DI A: "; A
RUN
```

Output atteso:

```
VALORE DI A: 5
```

Esempio 3 – Uso del punto e virgola ;
→ Evita il ritorno a capo

```
10 PRINT "A = ";
20 PRINT 10
RUN
```

Output atteso:

```
A = 10
```

Esempio 4 – Stampa su colonne con virgola ,

```
10 PRINT "X", "Y", "Z"
RUN
```

Output atteso:

```
X       Y       Z
```

Esempio 5 – Stampa di espressioni

```
10 PRINT "2 + 3 = "; 2 + 3
RUN
```

Output atteso:

```
2 + 3 = 5
```

Esempio 6 – Stampa di stringhe concatenate

```
10 A$ = "LUCA"
20 PRINT "CIAO " + A$
RUN
```

Output atteso:

```
CIAO LUCA
```

Nota:

- PRINT può essere abbreviato con ? (es: ? "CIAO")
- Per andare a capo, usa PRINT senza argomenti

PRINTUSING

Sintassi:

PRINTUSING "maschera"; espressione

Descrizione:

Il comando **PRINTUSING** stampa il risultato di una **espressione** usando una **maschera di formattazione** semplice.

Nel tuo interprete:

- La **maschera** è una stringa tra virgolette, ad esempio "0.00", "###.###", "VALORE: 0.000".
- Dalla maschera viene ricavato **solo il numero di decimali** (contando le cifre dopo il punto . o la virgola ,).
- Se l'espressione è **numerica**, il valore viene stampato con quel numero di decimali.
- Se l'espressione è **stringa**, la maschera viene ignorata e la stringa viene stampata così com'è.
- Alla fine, viene sempre mandato un **a capo** (equivalente a PRINT da solo).

Non gestisce:

- più valori sulla stessa PRINTUSING
- allineamenti complessi
- riempimenti con #, +, *, ecc.

È una **PRINT formattata semplificata** per fissare i decimali.

Esempi pratici

Esempio 1 – Numero con due decimali

```
10 X = 12.3456
20 PRINTUSING "0.00"; X
RUN
```

Output atteso:

12.35

(la maschera ha **2 cifre decimali**, quindi stampa con 2 decimali)

Esempio 2 – Numero con tre decimali

```
10 PI = 3.14159
20 PRINTUSING "###.000"; PI
RUN
```

Output atteso:

```
3.142
```

(la maschera ha **3 cifre decimali** dopo il punto)

Esempio 3 – Testo + numero formattato

```
10 VALORE = 7.5
20 PRINT "RISULTATO:"
30 PRINTUSING "0.0000"; VALORE
RUN
```

Output atteso:

```
RISULTATO:
7.5000
```

Esempio 4 – Maschera con testo fisso (gestito dal programma)
Dato che PRINTUSING formatta solo il numero, puoi costruire il testo prima:

```
10 VALORE = 123.456
20 PRINTUSING "VALORE = 0.00"; VALORE
RUN
```

Output atteso:

```
VALORE = 123.46
```

Esempio 5 – Espressione invece di variabile

```
10 PRINTUSING "0.000"; 10 / 3
RUN
```

Output atteso:

```
3.333
```

Esempio 6 – Espressione stringa
Se l’espressione è stringa, la maschera è ignorata e viene stampata la stringa:

```
10 S$ = "CIAO"
20 PRINTUSING "0.00"; S$
RUN
```

Output atteso:

```
CIAO
```

Nota:

- La maschera deve essere una stringa tra virgolette dopo PRINTUSING.
- Dalla maschera viene usato solo il numero di cifre decimali per formattare i numeri.
- Se il valore non è numerico, viene stampato come stringa, senza formattazione.
- Dopo ogni PRINTUSING viene emesso un CRLF (nuova riga), come PRINT.

READ

Sintassi:

READ variabile

Descrizione:
Il comando READ preleva un valore da una sequenza definita da uno o più comandi DATA.
È usato per **caricare dati predefiniti** nel programma in modo ordinato e sequenziale.
Ogni chiamata a READ estrae il **valore successivo** dalla lista DATA.

I valori DATA possono essere numeri o stringhe, separati da virgole.
Per **reinizializzare** la lettura dei dati, si usa RESTORE.

Esempi pratici

Esempio 1 – Lettura di numeri da DATA

```
10 READ A
20 READ B
30 PRINT A + B
40 DATA 3, 7
RUN
```

Output atteso:

```
10
```

Esempio 2 – Lettura di stringhe

```
10 READ NOME$
20 PRINT "CIAO "; NOME$
30 DATA "Luca"
RUN
```

Output atteso:

```
CIAO Luca
```

Esempio 3 – Lettura in un ciclo
→ Carica più valori da un blocco DATA

```
10 FOR I = 1 TO 3
20 READ X
30 PRINT "VALORE "; I; ": "; X
40 NEXT I
50 DATA 10, 20, 30
RUN
```

Output atteso:

VALORE 1: 10
VALORE 2: 20
VALORE 3: 30

Esempio 4 – Uso di RESTORE per riavviare la lettura

```
10 READ A
20 READ B
30 PRINT A, B
40 RESTORE
50 READ C
60 PRINT C
70 DATA 1, 2
RUN
```

Output atteso:

```
1       2
1
```

Esempio 5 – Errore se mancano valori DATA

→ Se ci sono più READ che dati, il programma può dare errore o comportamento imprevisto.

Nota:

- I comandi READ leggono sempre nell'ordine definito dai DATA
- È possibile posizionare DATA **in qualunque riga**, anche dopo READ

REBOOT

Sintassi:

REBOOT

Descrizione:
Il comando REBOOT forza il **riavvio completo del microcontrollore ESP32**.
È utile per:

- Ripristinare lo stato iniziale
- Applicare modifiche permanenti
- Uscire da condizioni di errore
- Simulare un "power reset" da software

Viene eseguito **immediatamente** e interrompe ogni programma in corso.

Esempi pratici

Esempio 1 – Riavvio dopo conferma

```
10 INPUT "SEI SICURO DI RIAVVIARE? (1 = SI) "; A
20 IF A = 1 THEN REBOOT
RUN
```

Output atteso:
Se l'utente inserisce 1, l'ESP32 si riavvia.

Esempio 2 – Uso in un programma di installazione

```
10 PRINT "INSTALLAZIONE COMPLETATA"
20 PRINT "RIAVVIO TRA 5 SECONDI..."
30 DELAY 5000
40 REBOOT
RUN
```

Output atteso:
Messaggio seguito da riavvio automatico.

Nota:

- Nessun salvataggio automatico viene eseguito: salvare prima eventuali dati importanti
- Il comando REBOOT non ha parametri e non restituisce alcun output

RENAME

(o **RENAME F$ N$**)

Sintassi

RENAME "vecchio" "nuovo"
RENAME variabile1$ variabile2$
RENAME SD "vecchio" "nuovo"
RENAME SD variabile1$ variabile2$
RENAME SPIFFS "vecchio" "nuovo"
RENAME SPIFFS variabile1$ variabile2$

Descrizione

Il comando **RENAME** rinomina un file sia nella **memoria interna SPIFFS** che sulla **scheda SD**, a seconda dell'argomento specificato.

- Senza argomenti di memoria, agisce su **SPIFFS** (memoria interna).
- Con SD, rinomina i file presenti sulla **scheda SD**.
- Accetta sia **nomi diretti tra virgolette**, sia **variabili stringa** o **espressioni** (A$, LEFT$(N$,3)+".bas").
- Non modifica i contenuti dei file: è un'operazione solo sui **nomi**.

Esempi pratici

Esempio 1 – Rinomina su SPIFFS (default)

RENAME "boot.bas" "boot_old.bas"

Output atteso:

File renamed on SPIFFS.

Esempio 2 – Rinomina su SD

RENAME SD "log.txt" "log_old.txt"

Output atteso:

File renamed on SD.

Esempio 3 – Con variabili stringa

10 F$ = "data.csv"
20 N$ = "data_2025.csv"

30 RENAME SD F$ N$
RUN

Output atteso:

File renamed on SD.

Esempio 4 – File inesistente

RENAME "missing.txt" "whatever.txt"

Output atteso:

FILE RENAME: Rename failed on SPIFFS.

Note

- Se nessuna memoria è specificata, l’operazione viene eseguita su **SPIFFS**.
- Il comando **non crea cartelle** e **non sposta** file tra directory diverse: rinomina solo all’interno del percorso indicato.
- I nomi dei file sono **case-sensitive** a seconda del file system.
- In caso di errore (file mancante, permessi, nome invalido), viene mostrato FILE RENAME: Rename failed.

RESTORE

Sintassi:

```
RESTORE
```

Descrizione:

Il comando RESTORE **riporta il puntatore dei READ all'inizio dei DATA**, permettendo di leggere nuovamente i dati dall'inizio.
È utile quando vuoi **riutilizzare gli stessi dati** in un secondo ciclo di lettura.

RESTORE **non prende argomenti** e agisce sempre su tutti i DATA, ricominciando dalla prima voce disponibile.

Esempi pratici

Esempio 1 – Lettura e ripetizione dei dati

```
10 READ A
20 READ B
30 PRINT "PRIMA LETTURA:", A, B
40 RESTORE
50 READ X
60 PRINT "DOPO RESTORE:", X
70 DATA 5, 10
RUN
```

Output atteso:

```
PRIMA LETTURA: 5     10
DOPO RESTORE: 5
```

Esempio 2 – Uso in un ciclo per leggere due volte la stessa sequenza

```
10 FOR I = 1 TO 2
20 RESTORE
30 READ A, B
40 PRINT "CICLO"; I; ": "; A; B
50 NEXT I
60 DATA 1, 2
RUN
```

Output atteso:

```
CICLO1: 1     2
CICLO2: 1     2
```

Esempio 3 – Combinazione con FOR...NEXT
→ Riutilizzo dei dati da capo:

```
10 FOR I = 1 TO 3
20 READ X
30 PRINT "X ="; X
40 NEXT I
50 RESTORE
60 READ Y
70 PRINT "Y dopo RESTORE ="; Y
80 DATA 10, 20, 30
RUN
```

Output atteso:

```
X = 10
X = 20
X = 30
Y dopo RESTORE = 10
```

Esempio 4 – Lettura errata senza RESTORE
→ Dopo la prima lettura, i dati sono esauriti:

```
10 READ A
20 READ B
30 READ C
40 PRINT A, B, C
50 DATA 1, 2
RUN
```

Output atteso:
Errore o valore imprevisto, perché DATA ha solo 2 elementi.

Nota:

- Se hai più blocchi DATA in diverse righe, RESTORE **ricomincia dalla prima disponibile**
- Non puoi puntare a una posizione intermedia (non esiste RESTORE n)

RETURN

Sintassi:

RETURN

Descrizione:
Il comando RETURN segnala la **fine di una subroutine** avviata con GOSUB.
Quando viene eseguito, il programma **torna alla riga immediatamente successiva** a quella da cui era stato chiamato GOSUB.

Ogni GOSUB **deve avere un corrispondente RETURN**, altrimenti il programma non ritorna correttamente al flusso principale.

Esempi pratici

Esempio 1 – Subroutine semplice

```
10 GOSUB 100
20 PRINT "RITORNO AL MAIN"
30 END
100 PRINT "SUBROUTINE"
110 RETURN
RUN
```

Output atteso:

```
SUBROUTINE
RITORNO AL MAIN
```

Esempio 2 – Uso con parametri indiretti (variabili globali)
→ Passaggio implicito di dati:

```
10 A = 5: B = 7
20 GOSUB 100
30 PRINT "SOMMA: "; R
40 END
100 R = A + B
110 RETURN
RUN
```

Output atteso:

```
SOMMA: 12
```

Esempio 3 – Uso con ON x GOSUB
→ Esecuzione condizionata di subroutine:

```
10 INPUT X
20 ON X GOSUB 100, 200
30 PRINT "FINE"
40 END
100 PRINT "SCELTA 1": RETURN
200 PRINT "SCELTA 2": RETURN
RUN
```

Output atteso (es. input 1):

```
SCELTA 1
FINE
```

Esempio 4 – Errore se manca RETURN
→ Il programma **non torna** se RETURN è assente:

```
10 GOSUB 100
20 PRINT "QUESTA NON VIENE ESEGUITA"
100 PRINT "MANCATO RETURN"
RUN
```

Output atteso:

```
MANCATO RETURN
```

(e poi blocco o comportamento inatteso)

Nota:

- Le subroutine possono essere **annidate**, ma ogni GOSUB deve terminare con un RETURN
- Può esserci più di un RETURN all'interno di condizioni (IF) per subroutine flessibili

RFID HALT

Sintassi:

RFID HALT

Descrizione:
Interrompe la comunicazione con il tag attualmente selezionato, liberandolo per nuove letture.
È buona pratica chiamarlo **dopo** aver letto o scritto un tag.

Esempi pratici

Esempio 1 – Leggi UID e rilascia:

```
10 RFID INIT 5 27
20 RFID WAITUID U$
30 PRINT "Trovato UID=";U$
40 RFID HALT
RUN
Output atteso:
Trovato UID=04A1B2C3D4
```

Esempio 2 – Uso in un ciclo:

```
10 RFID INIT 5 27
20 FOR I=1 TO 3
30   PRINT "Avvicina un tag..."
40   RFID WAITUID U$
50   PRINT "UID=";U$
60   RFID HALT
70 NEXT I
RUN
Output atteso:
Avvicina un tag...
UID=1122334455
Avvicina un tag...
UID=8899AABBCC
...
```

Note:
• Non è obbligatorio, ma utile per evitare letture multiple dello stesso tag senza rimuoverlo.
• Chiamalo prima di aspettare un nuovo tag (WAITUID).

RFID INIT (MFRC522)

Sintassi

RFID INIT ss rst
RFID INIT ss rst mosi miso sck

Descrizione

Inizializza il modulo RFID **MFRC522**.

Parametri:

- ss → pin SDA / SS del modulo RFID
- rst → pin RST del modulo

Parametri opzionali SPI:

- mosi, miso, sck

Se i pin SPI non sono specificati, vengono usati i **default ESP32**:

- MOSI = 23
- MISO = 19
- SCK = 18

Esempi

Esempio 1 – Inizializzazione base

```
10 RFID INIT 5 27
20 PRINT "RFID pronto"
RUN
```

Esempio 2 – Inizializzazione con pin SPI personalizzati

```
10 RFID INIT 5 27 23 19 18
20 PRINT "RFID pronto"
RUN
```

Esempio 3 – Con altri dispositivi SPI

```
10 PINMODE 17 OUTPUT NOPULL   ' CS display
20 DWRITE 17 1
30 PINMODE 4 OUTPUT NOPULL    ' CS touch
40 DWRITE 4 1
50 RFID INIT 5 27
60 PRINT "RFID pronto"
RUN
```

Note

- Deve essere chiamato **una sola volta** all’avvio.
- Se non viene chiamato, tutti gli altri comandi RFID falliscono.
- Se condividi SPI, assicurati che gli altri CS siano HIGH.

RFID PRESENT

Sintassi:

RFID PRESENT

Descrizione:
Restituisce **1** se un tag è attualmente nel campo del lettore, altrimenti **0**.
Utile per verifiche rapide senza bloccare.

Esempi pratici

Esempio 1 – Stampa stato:

```
10 RFID INIT 5 27
20 PRINT RFID PRESENT
RUN
Output atteso:
1 (se presente), 0 (se assente)
```

Esempio 2 – Loop di polling:

```
10 RFID INIT 5 27
20 IF RFID PRESENT=1 THEN PRINT "Tag rilevato!"
30 GOTO 20
```

Note:
• È **non bloccante**, ottimo per cicli che fanno più cose contemporaneamente.

RFID READBLOCKABS

Sintassi:

RFID READBLOCKABS block var$ keyType keyHex

Descrizione:

Legge **16 byte** da un **blocco assoluto** di una card **MIFARE Classic**.
Risultato in var$ come stringa HEX (32 caratteri).

Esempi pratici

Esempio 1 – Lettura di un blocco:

```
10 RFID INIT 5 27
20 PRINT "Avvicina card"
30 RFID WAITUID U$
40 RFID READBLOCKABS 4 D$ A "FFFFFFFFFFFF"
50 PRINT "Blocco4=";D$
60 RFID HALT
RUN
Output atteso:
Blocco4=00112233445566778899AABBCCDDEEFF
```

Esempio 2 – Controlla dato in blocco:

```
10 RFID INIT 5 27
20 RFID WAITUID U$
30 RFID READBLOCKABS 4 DATA$ A "FFFFFFFFFFFF"
40 IF DATA$="CAFEBABE00000000000000000000000" THEN PRINT "TAG OK"
RUN
```

Note:

• block parte da 0; non usare i blocchi trailer (ogni 4° blocco).
• Necessario prima WAITUID o READUID.

RFID READUID

Sintassi:

RFID READUID var$

Descrizione:

Legge l'UID del tag **se presente in quel momento**, altrimenti var$="".
Non blocca il programma.

Esempi pratici

Esempio 1 – Polling UID:

```
10 RFID INIT 5 27
20 RFID READUID U$
30 IF U$<>"" THEN PRINT "UID=";U$ ELSE PRINT "No Tag"
RUN
```

Esempio 2 – Usato in un ciclo temporizzato:

```
10 RFID INIT 5 27
20 FOR I=1 TO 10
30   RFID READUID U$
40   IF U$<>"" THEN PRINT "Trovato ";U$
50   WAIT 500
60 NEXT I
```

Note:

• Non sostituisce WAITUID se vuoi aspettare il tag.

RFID WAITUID

Sintassi:

RFID WAITUID var$ [timeout_ms] [GOTO line]

Descrizione:
Attende che venga presentato un tag e mette l'UID in var$.

- Se timeout_ms scade **e c'è GOTO line**, non dà errore: **salta** alla riga indicata.
- Se timeout_ms scade **senza GOTO**, non dà errore: var$ rimane "" e il programma **prosegue**.

Esempi pratici

Esempio 1 – Attesa indefinita

```
10 RFID INIT 5 27
20 PRINT "Avvicina un tag..."
30 RFID WAITUID U$
40 PRINT "UID=";U$
RUN
```

Esempio 2 – Timeout di 3s (gestito in IF)

```
10 RFID INIT 5 27
20 RFID WAITUID U$ 3000
30 IF U$="" THEN PRINT "Timeout" ELSE PRINT "UID=";U$
RUN
```

Esempio 3 – Timeout di 3s con salto

```
10 RFID INIT 5 27
20 PRINT "Presenta un tag (3s)..."
30 RFID WAITUID U$ 3000 GOTO 80
40 PRINT "UID=";U$
50 GOTO 100
80 PRINT "Nessun tag entro il tempo"
100 END
RUN
```

Note:
• Blocca finché non arriva un tag o scade il timeout.
• GOTO è **opzionale** e si applica solo al **caso di timeout**.
• var$ è una variabile stringa (deve terminare con $).

RFID WRITEBLOCKABS

Sintassi:

RFID WRITEBLOCKABS block "dataHex32" keyType keyHex

Descrizione:

Scrive **16 byte** (32 HEX) nel blocco assoluto block di una card MIFARE Classic.

Esempi pratici

Esempio 1 – Scrittura:

```
10 RFID INIT 5 27
20 PRINT "Avvicina card per scrittura"
30 RFID WAITUID U$
40 RFID WRITEBLOCKABS 4 "11223344556677889900AABBCCDDEEFF" A "FFFFFFFFFFFF"
50 PRINT "Scritto"
RUN
```

Esempio 2 – Scrivi stringa codificata:

```
10 DATA$="HELLO WORLD     "
20 HEX$=TOHEX(DATA$) ' ipotetico comando di conversione
30 RFID WRITEBLOCKABS 4 HEX$ A "FFFFFFFFFFFF"
```

Note:

• Pericoloso: può bloccare la card se scrivi i trailer.
• Verifica sempre il blocco e i permessi.

RFIDDB ADD

Sintassi:

```
RFIDDB ADD uid ["label"|label$]
RFIDDB ADD PRESENT ["label"|label$] [timeout_ms] [GOTO line]
RFIDDB ADD SCAN    ["label"|label$] [timeout_ms] [GOTO line]
```

Descrizione:

Aggiunge o aggiorna un tag nel DB **in RAM**.

- uid può essere **stringa** HEX o **variabile stringa** (U$).
- Con PRESENT/SCAN il sistema attende un tag: se letto entro timeout_ms, lo aggiunge; altrimenti:
  - con GOTO line → **salta** a line;
  - senza GOTO → **non** dà errore, semplicemente non aggiunge nulla e prosegue.
- label è opzionale e può essere **stringa** o **variabile stringa** (N$).

Esempi pratici

Esempio 1 – Inserimento manuale (UID e label letterali)

```
10 RFIDDB INIT "tags.json"
20 RFIDDB ADD “04A1B2C3D4” "Mario"
30 RFIDDB SAVE
RUN
```

Esempio 2 – Inserimento manuale con variabili

```
10 RFIDDB INIT "tags.json"
20 U$="04A1B2C3D4"
30 N$="Mario"
40 RFIDDB ADD U$ N$
50 RFIDDB SAVE
RUN
```

Esempio 3 – PRESENT con label e timeout (senza salto)

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Avvicina un nuovo tag (5s)"
40 RFIDDB ADD PRESENT "Alice" 5000
50 RFIDDB SAVE
RUN
```

Esempio 4 – PRESENT con label in variabile e salto su timeout

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 N$="Ospite"
40 PRINT "Avvicina un nuovo tag (5s)"
50 RFIDDB ADD PRESENT N$ 5000 GOTO 120
60 PRINT "Tag aggiunto!"
```

```
70 RFIDDB SAVE
80 RFIDDB LIST
90 GOTO 200
120 PRINT "Tempo scaduto: nessun tag aggiunto"
200 END
RUN
```

Note:

- Modifica solo la **RAM** → usa RFIDDB SAVE per scrivere su SPIFFS.
- uid e label accettano **variabili stringa** (es. U$, N$) o **stringhe** tra virgolette.
- PRESENT e SCAN sono equivalenti come comportamento di acquisizione UID.

RFIDDB CLEAR

Sintassi:

RFIDDB CLEAR

Descrizione:
Svuota il DB in RAM (non tocca il file su SPIFFS).

Esempio:

```
10 RFIDDB INIT "tags.json"
20 RFIDDB CLEAR
30 PRINT "Totale=";RFIDDB COUNT
RUN
Output atteso:
Totale=0
```

Note:
• Per pulizia permanente: CLEAR + SAVE.

RFIDDB COUNT

Sintassi:

RFIDDB COUNT

Descrizione:
Restituisce il numero di tag presenti in RAM.

Esempio:

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LOAD
30 PRINT "Ci sono ";RFIDDB COUNT;" tag"
```

RFIDDB DELETEFILE

Sintassi:

RFIDDB DELETEFILE

Descrizione:
Elimina il file JSON da SPIFFS. Il DB in RAM rimane intatto.

Esempio:

```
10 RFIDDB INIT "tags.json"
20 RFIDDB DELETEFILE
30 PRINT "Archivio cancellato"
```

RFIDDB EXISTS

Sintassi:

RFIDDB EXISTS uid

Descrizione:

Ritorna **1** se l'UID è in RAM, **0** altrimenti. Usabile in IF, PRINT, LET.

Esempio:

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 RFIDDB LOAD
40 RFID WAITUID U$
50 IF RFIDDB EXISTS U$ THEN PRINT "OK" ELSE PRINT "NO"
RUN
```

RFIDDB INIT

Sintassi

RFIDDB INIT "file.json"
RFIDDB INIT SD "file.json"

Descrizione

Seleziona lo storage e carica il database in RAM.

- Senza SD → SPIFFS
- Con SD → SD card

Esempi

10 RFIDDB INIT "tags.json"
10 RFIDDB INIT SD "tags.json"

Note

- Lo storage scelto resta valido per tutti i comandi RFIDDB successivi.

RFIDDB LABEL

Sintassi:

RFIDDB LABEL uid var$

Descrizione:

Restituisce in var$ l'etichetta associata a un UID.
Se l'UID non è presente, var$ sarà vuota.

Esempio pratico

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LABEL “04A1B2C3D4” NOME$
30 PRINT "Nome=";NOME$
RUN
```

Output atteso:
Nome=Mario

RFIDDB LIST

Sintassi:

RFIDDB LIST

Descrizione:
Stampa su seriale l’elenco di tutti i tag presenti in RAM, con UID ed etichetta.

Esempio pratico

10 RFIDDB INIT "tags.json"
20 RFIDDB LIST
RUN
Output atteso:
UID=04A1B2C3D4 LABEL="Mario"
UID=1122334455 LABEL="Alice"

RFIDDB LOAD

Sintassi:

RFIDDB LOAD

Descrizione:
Ricarica il database dal file JSON in SPIFFS, sostituendo il contenuto in RAM.

Esempio pratico

```
10 RFIDDB INIT "tags.json"
20 RFIDDB LOAD
30 PRINT "DB ricaricato"
RUN
```

RFIDDB REMOVE

Sintassi:

```
RFIDDB REMOVE uid
RFIDDB REMOVE PRESENT [timeout_ms] [GOTO line]
RFIDDB REMOVE SCAN    [timeout_ms] [GOTO line]
```

Descrizione:
Rimuove un tag dal DB **in RAM**.

- Con uid: rimuove direttamente (accetta **stringa** o **variabile stringa** U$).
- Con PRESENT/SCAN: attende un tag; se letto entro timeout_ms, lo rimuove; se scade:
  - con GOTO line → **salta**;
  - senza GOTO → **non** dà errore, semplicemente non rimuove e prosegue.

Esempi pratici

Esempio 1 – Rimozione manuale (UID letterale)

```
10 RFIDDB INIT "tags.json"
20 RFIDDB REMOVE “04A1B2C3D4”
30 RFIDDB SAVE
RUN
```

Esempio 2 – Rimozione manuale con variabile

```
10 RFIDDB INIT "tags.json"
20 U$="04A1B2C3D4"
30 RFIDDB REMOVE U$
40 RFIDDB SAVE
RUN
```

Esempio 3 – Rimozione con PRESENT e timeout (senza salto)

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Avvicina tag da rimuovere (5s)"
40 RFIDDB REMOVE PRESENT 5000
50 RFIDDB SAVE
RUN
```

Esempio 4 – Rimozione con PRESENT e GOTO su timeout

```
10 RFID INIT 5 27
20 RFIDDB INIT "tags.json"
30 PRINT "Avvicina tag da rimuovere (5s)"
40 RFIDDB REMOVE PRESENT 5000 GOTO 120
50 PRINT "Rimosso!"
60 RFIDDB SAVE
70 GOTO 200
120 PRINT "Nessun tag: nulla rimosso"
200 END
RUN
```

Note:
• Modifica solo la **RAM** → usa RFIDDB SAVE per rendere permanente.
• uid accetta **stringa** o **variabile** (U$).
• PRESENT/SCAN sono equivalenti per la lettura dell'UID.
• Il salto GOTO avviene **solo** in caso di **timeout**.

RFIDDB SAVE

Sintassi:

RFIDDB SAVE

Descrizione:
Salva il database in RAM sul file JSON in SPIFFS.

Esempio pratico

```
10 RFIDDB INIT "tags.json"
20 RFIDDB ADD “04A1B2C3D4” "Mario"
30 RFIDDB SAVE
RUN
```

RIGHT$(A$, N)

Sintassi:

RIGHT$(stringa$, N)

Descrizione:
La funzione RIGHT$ restituisce una **sottostringa** contenente gli **ultimi N caratteri** della stringa A$.
Se N è maggiore della lunghezza della stringa, restituisce **l'intera stringa**.

È utile per:

- Estrarre estensioni di file
- Verificare suffissi
- Gestire codici, numeri finali, stringhe strutturate

Esempi pratici

Esempio 1 – Ultimi 3 caratteri

```
10 A$ = "BASIC32"
20 PRINT RIGHT$(A$, 3)
RUN
```

Output atteso:

```
C32
```

Esempio 2 – Estensione di un nome file

```
10 FILE$ = "config.bas"
20 EST$ = RIGHT$(FILE$, 4)
30 PRINT "ESTENSIONE: "; EST$
RUN
```

Output atteso:

```
ESTENSIONE: .bas
```

Esempio 3 – N maggiore della lunghezza

```
10 S$ = "OK"
20 PRINT RIGHT$(S$, 10)
RUN
```

Output atteso:

OK

Esempio 4 – Sottostringa vuota

```
10 T$ = "TEST"
20 PRINT RIGHT$(T$, 0)
RUN
```

Output atteso:

```
 (empty string)
```

Esempio 5 – Verifica se una stringa termina in ".BAS"

```
10 F$ = "AUTOEXEC.BAS"
20 IF RIGHT$(F$, 4) = ".BAS" THEN PRINT "FILE BASIC"
RUN
```

Output atteso:

```
FILE BASIC
```

Nota:

- N deve essere ≥ 0
- Funziona solo con variabili stringa ($)
- Combinabile con LEFT$, MID$, LEN, CHR$, ASC

RND(x) / RND(a, b)

Sintassi:

```
RND(x)        ' Numero casuale da 0 a x - 1
RND(a, b)     ' Numero casuale da a a b - 1
```

Descrizione:

La funzione RND genera **numeri interi casuali** secondo due modalità:

- ***RND(x) – Modalità base***

  - Restituisce un numero intero **tra 0 e x - 1**
  - Se x = -1 → genera sempre **lo stesso numero casuale** (seed fisso)
  - Se x = 0 → restituisce **l'ultimo numero casuale generato**

- ***RND(a, b) – Modalità estesa***

  - Restituisce un numero intero compreso tra **a** e **b - 1**
  - Valida anche con a > b (inverte automaticamente)
  - Utile per intervalli arbitrari (es: simulare un dado, scegliere da un range)

Esempi pratici

Esempio 1 – RND(10)

```
10 PRINT RND(10)
RUN
```

Output atteso:

Numero da 0 a 9 (es. 7)

Esempio 2 – RND(5, 15)

```
10 PRINT RND(5, 15)
RUN
```

Output atteso:

Numero da 5 a 14 (es. 12)

Esempio 3 – RND(-1)

→ Sempre lo stesso numero (utile per test/debug)

```
10 PRINT RND(-1)
RUN
```

Output atteso:

(Numero costante, es. 4)

Esempio 4 – RND(0)
→ Ultimo valore casuale generato

```
10 A = RND(10)
20 PRINT "Ultimo:", RND(0)
RUN
```

Output atteso:

Ultimo: (lo stesso numero della riga 10)

Esempio 5 – Simulare lancio di dado (1–6)

```
10 DADO = RND(1, 7)
20 PRINT "LANCIO: "; DADO
RUN
```

Output atteso:

LANCIO: 5

Esempio 6 – RND con parametri invertiti (valido)

```
10 PRINT RND(10, 5)
RUN
```

Output atteso:

Numero compreso tra 5 e 9

Riepilogo comportamenti

Forma	Risultato
RND(10)	Intero da **0 a 9**
RND(5, 15)	Intero da **5 a 14**
RND(-1)	Restituisce **sempre lo stesso numero**
RND(0)	Restituisce **l'ultimo numero generato**

Nota:

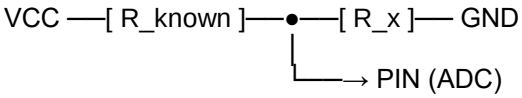
- Sempre restituito un **intero**
- Nessun bisogno di inizializzare il seme randomico
- Ottimo per menu, giochi, randomizzazione generica

RREAD

Sintassi:

RREAD <pin> <VCC> <R_known>

Descrizione:
Calcola una **resistenza incognita** R_x misurando la tensione sul nodo del partitore nello schema:



Formula:
R_x = (Vnode * R_known) / (VCC - Vnode)
dove Vnode è la tensione letta al pin (con ADC CAL applicata).

Esempi pratici

Esempio 1 – R_known = 10k, VCC = 3.3V, pin 34

```
10 RX = RREAD 34 3.3 10000
20 PRINT "Rx=", RX, " ohm"
```

Esempio 2 – Con variabili

```
10 PIN=34: VCC=3.3: RK=10000
20 RX = RREAD PIN VCC RK
30 PRINT "Rx=", RX
```

Esempio 3 – Debug rapido del nodo

```
10 VP = VREAD 34
20 PRINT "Nodo (V)=", VP
30 RX = RREAD 34 3.3 10000
40 PRINT "Rx=", RX
```

Output atteso:

```
Rx= 9873.42  ohm
```

In caso di nodo a 0 V (corto a GND o partitore errato):

```
?RREAD ERROR
vout=0 (check wiring)
```

In caso di R_known non valido:

```
?RREAD ERROR
```

r_known must be > 0

Nota:

- Schema **obbligatorio**: R_known in alto verso VCC, R_x verso **GND**, pin ADC al nodo centrale.
- VCC deve essere la reale alimentazione del partitore (es. 3.3).
- RREAD usa la calibrazione di ADC CAL.
- Se stai usando lo **schema inverso** (R_x in alto, R_known in basso) la formula cambia; possiamo aggiungere un comando separato se ti serve.

RUN

Sintassi:

RUN
RUN FAST
RUN "nomefile.bas"
RUN SPIFFS "nomefile.bas"
RUN SD "nomefile.bas"

Descrizione

RUN avvia l'esecuzione del programma BASIC in memoria, oppure carica ed esegue un file .bas da SPIFFS o SD.

Funzionamento standard

- Se usato **senza parametri**, esegue il listato attualmente in memoria.
- Se usato con un nome file tra virgolette:
  - carica il file dalla memoria predefinita (SPIFFS o SD)
  - e lo esegue.
- Specificando **SPIFFS** o **SD**, si forza la sorgente del file.

RUN FAST (nuovo)

RUN FAST esegue il programma alla **massima velocità possibile**, disabilitando temporaneamente:

- polling tastiere (PS/2 e CardKB)
- cattura tasti ESC
- eco della console
- controlli di input interattivo

È ideale per:

- giochi con cicli pesanti
- benchmark
- grafica ad alta velocità
- loop intensivi

Durante RUN FAST, l'ingresso tastiera non funziona, a meno che non venga riattivato dal programma tramite KEYPOLL ON.

Comportamento di RUN:

- Interrompe qualsiasi programma in corso.
- Può essere richiamato dopo STOP, dopo un errore oppure da un altro programma.

- Se un file .bas viene caricato tramite RUN, il listato attuale viene sovrascritto.
- Se RUN FAST viene eseguito, il polling tastiere parte disattivato.

Esempi pratici

Esempio 1 — Eseguire il programma in memoria

```
10 PRINT "CIAO MONDO"
20 END
RUN
```

Output:

```
CIAO MONDO
```

Esempio 2 — Riavvio dopo STOP

```
10 INPUT X
20 IF X = 0 THEN STOP
30 PRINT "VALORE: "; X
```

Esecuzione:

```
? 0
(STOP)
```

Poi:

```
RUN
```

Il programma riparte da capo.

Esempio 3 — Eseguire un file da SPIFFS

```
RUN "menu.bas"
```

Carica ed esegue menu.bas dalla sorgente predefinita.

Esempio 4 — Forzare sorgente specifica

```
RUN SPIFFS "demo.bas"
RUN SD "gioco.bas"
```

Esempio 5 — RUN FAST

RUN FAST

Esegue il programma in modalità turbo (nessun polling tastiera).

Note

- Se il file non esiste → errore.
- Se la memoria è vuota e si esegue RUN, il sistema non fa nulla.
- RUN FAST non disabilita il BREAKPIN.
- Durante RUN FAST si può riabilitare la tastiera da codice con KEYPOLL ON.

SAVE

(o **SAVE F$**)

Sintassi:

SAVE "nomefile"
SAVE variabile$
SAVE SD "nomefile"
SAVE SD variabile$
SAVE SPIFFS "nomefile"
SAVE SPIFFS variabile$

Descrizione:

Il comando **SAVE** salva il programma BASIC attualmente in memoria su file.

- Senza specificare la memoria, il salvataggio avviene nella **memoria interna SPIFFS**.
- Specificando **SD**, il file viene salvato sulla **scheda SD**.
- È possibile indicare anche **SPIFFS** per chiarezza, ma non è obbligatorio.
- Il nome del file può essere indicato **direttamente tra virgolette** oppure tramite **una variabile stringa** (es. F$).

Il file viene salvato in formato testuale, con le linee numerate come nel listato BASIC.
Se un file con lo stesso nome è già presente, viene **sovrascritto**.
L'estensione consigliata è .bas.

Esempi pratici

Esempio 1 – Salvare nella memoria interna SPIFFS

SAVE "config.bas"

Output atteso:
Il file *config.bas* viene salvato nella memoria interna, anche se è presente una scheda SD.

Esempio 2 – Salvare su scheda SD

SAVE SD "programma.bas"

Output atteso:
Il file *programma.bas* viene salvato sulla scheda SD.

Esempio 3 – Usare una variabile stringa per il nome del file

```
10 LET F$ = "menu.bas"
20 SAVE F$
RUN
```

Output atteso:
Il file *menu.bas* viene salvato nella memoria interna SPIFFS.

Esempio 4 – Salvare su SD usando una variabile

```
10 LET F$ = "demo.bas"
20 SAVE SD F$
RUN
```

Output atteso:
Il file *demo.bas* viene salvato sulla scheda SD.

Nota:

- Se la scheda SD non è collegata o non è inizializzata, SAVE SD ... restituirà un errore.
- Se il nome del file non termina con .bas, il sistema suggerirà automaticamente di aggiungerla.
- Il comando SAVE **sovrascrive** eventuali file già presenti con lo stesso nome.
- È possibile utilizzare **variabili stringa** per specificare dinamicamente il nome del file.

SAVEVAR

Nome

SAVEVAR

Sintassi

SAVEVAR "file" "chiave" valore
SAVEVAR SPIFFS "file" "chiave" valore
SAVEVAR SD "file" "chiave" valore

Descrizione

SAVEVAR salva (o aggiorna) una variabile dentro un file **JSON**.

- Se il file non esiste, viene creato.
- Se la chiave esiste già, viene **sovrascritta**.
- Il valore può essere:
  - **numero** (es. 150, 22.5, oppure una variabile numerica come A)
  - **stringa** (es. "Anna", oppure una variabile stringa come U$)

Il file viene salvato nel filesystem selezionato:

- **default** se non specifichi niente
- **SPIFFS** se scrivi SPIFFS
- **SD** se scrivi SD

Esempi

Esempio 1 – Salvare configurazione su SPIFFS

```
10 SAVEVAR SPIFFS "prefs.json" "SPEED" 150
20 SAVEVAR SPIFFS "prefs.json" "USER" "Anna"
```

Risultato (contenuto tipico del file):

```
{
  "SPEED": 150,
  "USER": "Anna"
}
```

Esempio 2 – Salvare dati su SD

```
10 SAVEVAR SD "dati.json" "TEMP" 22.5
20 SAVEVAR SD "dati.json" "STATO" "OK"
```

Esempio 3 – Salvare usando variabili BASIC

```
10 A = 42
20 N$ = "Mario"
30 SAVEVAR "test.json" "NUM" A
40 SAVEVAR "test.json" "NOME" N$
```

Note

- Le stringhe **devono** essere tra virgolette: "testo".
- Se salvi su SD, la SD deve essere inizializzata/montata (altrimenti errore runtime).
- SAVEVAR mantiene le altre chiavi già presenti nel file: aggiorna solo quella indicata.

SCANI2C

Sintassi:

SCANI2C

Descrizione:

Il comando **SCANI2C** esegue una **scansione del bus I2C** e stampa nel monitor seriale tutti gli indirizzi dei dispositivi rilevati.

 Prima di usare SCANI2C è obbligatorio chiamare il comando WIRE per impostare i pin SDA e SCL del bus I2C.
Se WIRE non è stato eseguito, viene generato un errore a runtime.

Esempi pratici

Esempio 1 – Scansione semplice del bus

10 WIRE 21 22
20 SCANI2C
RUN

Output tipico:

Scanning I2C bus...
Found device at 0x3C (60)
Total I2C devices: 1

Esempio 2 – Errore se WIRE manca

10 SCANI2C
RUN

Output atteso:

RUNTIME ERROR: SCANI2C: WIRE must be called first

Note:

- Utile per verificare se i sensori I2C sono correttamente collegati.
- Mostra gli indirizzi in esadecimale (es. 0x3C) e in decimale (es. 60).
- Il risultato appare nel **monitor seriale**.
- Assicurati di aver impostato correttamente i pin con WIRE sda scl prima di usarlo.

SDFREE

Sintassi:
PRINT SDFREE

Descrizione:
Il comando SDFREE restituisce lo spazio libero disponibile sulla scheda SD.
Serve per conoscere la memoria residua prima di effettuare salvataggi o letture da file.

Esempi pratici

Esempio 1 – Mostrare spazio disponibile su SD

```
10 PRINT SDFREE
RUN
```

Output atteso:

```
spazio disponibile
```

Nota:
• Il valore è espresso in byte
• Funziona solo se la SD è montata correttamente

SERVOATTACH

Sintassi:
SERVOATTACH <id> <pin> [minUS] [maxUS]

Descrizione:
Collega un servo all'ID indicato sul pin scelto.
• Frequenza: **50 Hz**.
• Range impulsi: se specificato viene *clampato* a **300..3000 µs**.
• Fallback automatici se l'attach fallisce: prima 500..2400, poi 1000..2000.
• Se l'ID era su un altro pin, fa detach e ri-attacca sul nuovo.
• Rifiuta i pin **34..39** (input-only).

Esempi pratici
Esempio 1 – Attach base

```
10 SERVOMAX 1
20 SERVOATTACH 0 25
30 SERVOSTATUS
```

⟶ Attacca l'ID 0 al pin 25 con range predefinito (500..2400).

Esempio 2 – Range personalizzato

```
10 SERVOATTACH 0 25 600 2400
20 SERVOWRITEMICROS 0 1500
```

⟶ Usa un range più ampio lato minimo; i comandi in µs verranno clampati a 600..2400.

Output atteso:
Nessuna stampa se ok. In errore, messaggi del tipo:
SYNTAX SERVOATTACH id pin [minUS] [maxUS]
VALUE id out of cap / VALUE id >= SERVOMAX
VALUE Invalid pin / VALUE Pin is input-only on ESP32
VALUE maxUS must be > minUS
HW attach failed

Nota:
• Pin consigliati: 25, 26, 27, 32, 33.
• Il range **effettivo** usato (dopo i fallback) viene memorizzato e impiegato dai comandi in µs.

SERVODETACH

Sintassi:
SERVODETACH <id>

Descrizione:
Scollega il servo dall'ID indicato e libera le risorse associate.

Esempi pratici
Esempio 1 – Detach al termine

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 90
30 WAIT 1000
40 SERVODETACH 0
```

Esempio 2 – Detach multipli

```
10 SERVOMAX 3
20 FOR I=0 TO 2
30  SERVODETACH I
40 NEXT I
```

Output atteso:
Nessuna stampa. In errore:
VALUE Invalid id or >= SERVOMAX

Nota:
• Dopo il detach, il pin resta nello stato impostato; se serve, ripristinalo tu (es. PINMODE / DIGITALWRITE nella tua shell BASIC se disponibili).

SERVOMAX

Sintassi:
SERVOMAX <n>

Descrizione:
Imposta quanti servo sono gestibili a runtime.
• Valori ammessi: 1 … MAX_BASIC_SERVOS_CAP (cap = 16).
• Se riduci n, gli ID ≥ n vengono automaticamente **detach**.

Esempi pratici
Esempio 1 – Limitare a 4 servo

```
10 SERVOMAX 4
20 SERVOSTATUS
```

⟶ Imposta il limite runtime a 4; “SERVOSTATUS” riflette il nuovo valore.

Esempio 2 – Riduzione con detach automatico

```
10 SERVOMAX 4
20 SERVOATTACH 3 27
30 SERVOATTACH 4 32
40 SERVOMAX 4
50 SERVOSTATUS
```

⟶ L’ID 4 viene detach perché oltre il nuovo limite.

Output atteso:
Nessuna stampa se non usi SERVOSTATUS.

Nota:
• Default tipico: BASIC_SERVO_MAX = 8.
• Riduci SERVOMAX se hai pochi servo per risparmiare risorse.

SERVOREAD

Sintassi:
SERVOREAD <id>

Descrizione:
Restituisce l'**angolo logico** corrente del servo (quello impostato via SERVOWRITE), oppure **-1** se l'ID non è valido o il servo non è attaccato.
• Usabile dentro espressioni/assegnazioni.

Esempi pratici
Esempio 1 – Lettura semplice

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 90
30 A = SERVOREAD 0
40 PRINT "ANGLE=", A
```

Esempio 2 – Controllo condizionale

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 45
30 IF SERVOREAD 0 < 90 THEN PRINT "OK"
```

Output atteso:
ANGLE= 90 (nell'esempio 1)

Nota:
• È un valore di **stato interno**, non una misura fisica dell'albero del servo.

SERVOSTATUS

Sintassi:
SERVOSTATUS

Descrizione:
Stampa lo stato del sottosistema LEDC/servo:
• Timer LEDC allocati (YES/NO).
• BASIC_SERVO_MAX e MAX_BASIC_SERVOS_CAP.
• Per ogni ID: stato (ATTACHED/detached), PIN e RANGE[us]=min..max.
• Conteggio dei servo attaccati e note/pin consigliati.

Esempi pratici
Esempio 1 – Stato iniziale

```
10 SERVOSTATUS
```

⟶ Mostra configurazione corrente, utile per il debug prima di usare i servo.

Esempio 2 – Dopo attach

```
10 SERVOMAX 2
20 SERVOATTACH 0 25 500 2400
30 SERVOATTACH 1 26
40 SERVOSTATUS
```

⟶ Verifica che gli ID 0 e 1 siano “ATTACHED” con PIN e range.

Output atteso (esempio indicativo):

```
=== SERVO STATUS ===
Timers allocated: YES
BASIC_SERVO_MAX: 2  (CAP=16)
ID 0: ATTACHED  PIN=25  RANGE[us]=500..2400
ID 1: ATTACHED  PIN=26  RANGE[us]=500..2400
ID 2: detached  PIN=-1  RANGE[us]=0..0
...
Attached count: 2
Note:
- ...
============================
```

Nota:
• Utile per individuare conflitti LEDC o pin non adatti.
• I pin consigliati sono: 25, 26, 27, 32, 33. Evita 34..39 (input-only) e i pin SPI (5, 13, 14, 15, 18, 19, 23).

SERVOWRITE

Sintassi:
SERVOWRITE <id> <angle>

Descrizione:
Imposta l'angolo di un servo in **gradi**.
• L'angolo viene saturato a **0..180**.
• Richiede che il servo sia **attach**.

Esempi pratici
Esempio 1 – Vai a 0°, poi 180°

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 0
30 WAIT 1000
40 SERVOWRITE 0 180
```

Esempio 2 – Tre posizioni

```
10 SERVOATTACH 0 25
20 SERVOWRITE 0 0
30 WAIT 500
40 SERVOWRITE 0 90
50 WAIT 500
60 SERVOWRITE 0 180
```

Output atteso:
Nessuna stampa. In errore:
VALUE Invalid id or >= SERVOMAX / STATE Servo not attached

Nota:
• Usa WAIT (ms) tra movimenti per dare tempo al servo di raggiungere la posizione.

SERVOWRITEMICROS

Sintassi:
SERVOWRITEMICROS <id> <micros>

Descrizione:
Imposta direttamente la larghezza impulso in **microsecondi**.
• Il valore viene *clampato* al range **effettivo** registrato in SERVOATTACH (es. 500..2400).
• Richiede che il servo sia **attach**.

Esempi pratici
Esempio 1 – Neutro tipico

```
10 SERVOATTACH 0 25
20 SERVOWRITEMICROS 0 1500
```

⟶ Impulso di 1.5 ms.

Esempio 2 – Con clamp automatico

```
10 SERVOATTACH 0 25 600 2400
20 SERVOWRITEMICROS 0 500
30 SERVOWRITEMICROS 0 2500
```

⟶ Il primo comando viene portato a 600 µs, il secondo a 2400 µs.

Output atteso:
Nessuna stampa. In errore:
VALUE Invalid id or >= SERVOMAX / STATE Servo not attached

Nota:
• Utile per calibrazioni fini o per servo a rotazione continua (dove 1500 µs ≈ stop).

SETDATE

Sintassi:
SETDATE anno mese giorno

Descrizione:
Il comando **SETDATE** imposta manualmente la **data** dell'orologio interno del sistema.
La stringa deve essere nel formato anno,mese,giorno.

Esempi pratici

Esempio 1 – Impostare la data al 15 giugno 2025

```
SETDATE 2025 6 15
PRINT DATED, DATEM, DATEY
```

Output atteso:

```
15 6 2025
```

Nota:
• Il formato deve essere esattamente "anno,mese,giorno"
• Non sincronizza l'orologio: è una modifica manuale
• Utile in assenza di rete o per configurare modulo RTC se presente
• Può essere usato insieme a SETTIME per impostazioni complete

SETTIME

Sintassi:
SETTIME ore minuti secondi

Descrizione:
Il comando **SETTIME** imposta manualmente l'**ora** dell'orologio interno del sistema.
La stringa deve essere nel formato ore,minuti,secondi (24 ore).

Esempi pratici

Esempio 1 – Impostare l'ora alle 14:38:00

```
10 SETTIME 14 38 00
20 PRINT TIMEH, TIMEM, TIMES
```

Output atteso:

```
14 38 00
```

Nota:
• Il formato deve essere esattamente "ore,minuti,secondi"
• Non sincronizza l'orologio: è una modifica manuale
• Utile in assenza di rete o per configurare modulo RTC se presente
• Consigliato usarlo in combinazione con SETDATE per impostazioni complete

SFXDEF

Sintassi:
SFXDEF name$ v wave$ pw atk dec sus rel

Descrizione:
Definisce un **effetto sonoro** (SFX) nominato e ne imposta timbro e inviluppo ADSR.
L'effetto viene memorizzato internamente e può essere riprodotto con SFXPLAY.

- name$ nome dell'effetto (stringa tra virgolette)
- v voce usata (consigliato **4**, voce SFX dedicata)
- wave$ "TRI", "SAW", "PULSE", "NOISE"
- pw pulse width (solo per "PULSE")
- atk dec sus rel parametri ADSR (0..15)

Esempio pratico – Definizione di uno sparo

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "SHOT" 4 "PULSE" 1200 0 3 0 2
40 GOTO 40
```

Note

- Ridefinire lo stesso name$ sovrascrive l'effetto.
- Non produce suono finché non viene chiamato con SFXPLAY.

SFXPLAY

Sintassi:
SFXPLAY name$

Descrizione:
Riproduce un effetto sonoro definito con SFXDEF.
Non interrompe la musica sulle voci 1..3.

Esempio pratico – Riproduzione SFX

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "COIN" 4 "TRI" 0 0 2 12 2
40 SFXSWEEP "COIN" "C6" "E6" 120
50 SFXPLAY "COIN"
60 GOTO 60
```

Note

- Se un altro SFX è già attivo sulla stessa voce, viene interrotto.
- Ideale per spari, salti, monete, batteria.

SFXSWEEP

Sintassi:
SFXSWEEP name$ start end dur

Descrizione:
Imposta uno **sweep di frequenza** (pitch che sale o scende) per un effetto.

- start, end note ("C7") o frequenze
- dur durata sweep in ms

Esempio pratico – Sparo discendente

```
10 SIDINIT 16000
20 SIDVOL 5
30 SFXDEF "SHOT" 4 "PULSE" 1200 0 3 0 2
40 SFXSWEEP "SHOT" "C7" "C4" 120
50 SFXPLAY "SHOT"
60 GOTO 60
```

Note

- Va chiamato dopo SFXDEF.
- Se non impostato, l'SFX usa frequenza fissa.

SIDINIT

Sintassi:
SIDINIT rate

Descrizione:

FUNZIONA SOLAMENTE SU ESP32 DEV!!!
Inizializza il motore audio SID-like in background e imposta il sample rate.

Per il funzionamento serve collegre al gpio25 dell’esp32 un amplificatore PAM8403 e un altoparlante.

Esempio pratico – Avvio audio

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "C4" 300
60 SIDMADD 2 "E4" 300
70 SIDMADD 2 "G4" 600
80 SIDMPLAY 0
90 GOTO 90
```

Note

- 16000 = suono più “C64 caldo”
- 22050 = più brillante

SIDMADD

Sintassi:
SIDMADD v noteOrHz dur

Descrizione:
Aggiunge un evento musicale alla sequenza.

- v **1..3** (solo voci musica)
- noteOrHz nota ("A4"), frequenza o "R" (pausa)
- dur durata in ms

Esempio pratico – Arpeggio platform

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 1 "TRI" 0 2 6 12 4
40 SIDVOICE 2 "PULSE" 900 3 6 10 5
50 SIDVOICE 3 "TRI" 0 3 6 8 5
60 SIDMCLR
70 SIDMADD 1 "C3" 200
80 SIDMADD 2 "E4" 200
90 SIDMADD 3 "G4" 200
100 SIDMPLAY 1
110 GOTO 110
```

Note

- SIDMADD 4 ... genera errore (voce riservata agli SFX).

SIDMCLR

Sintassi:
SIDMCLR

Descrizione:
Cancella la sequenza musicale caricata con SIDMADD.

Esempio pratico – Cambio pattern

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 1 "TRI" 0 2 6 12 4
40 SIDMCLR
50 SIDMADD 1 "C3" 300
60 SIDMADD 1 "G2" 300
70 SIDMPLAY 1
80 WAIT 800
90 SIDMCLR
100 SIDMADD 1 "F2" 300
110 SIDMADD 1 "C3" 600
120 SIDMPLAY 1
130 GOTO 130
```

Note

- Usalo prima di caricare una nuova musica.

SIDMPLAY

Sintassi:
SIDMPLAY loop

Descrizione:
Avvia la musica in background.

- loop = 1 ripete
- loop = 0 suona una volta

Esempio pratico – Jingle

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "PULSE" 900 2 5 9 4
40 SIDMCLR
50 SIDMADD 2 "C5" 150
60 SIDMADD 2 "E5" 150
70 SIDMADD 2 "G5" 300
80 SIDMPLAY 0
90 GOTO 90
```

Note

- Non blocca il programma BASIC.

SIDMSTOP

Sintassi:
SIDMSTOP

Descrizione:
Ferma la musica e chiude le note attive.

Esempio pratico – Stop musica

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "C4" 300
60 SIDMADD 2 "E4" 300
70 SIDMPLAY 1
80 WAIT 1000
90 SIDMSTOP
100 GOTO 100
```

Note

- Gli SFX possono continuare anche dopo SIDMSTOP.

SIDVOICE

Sintassi:
SIDVOICE v wave$ pw atk dec sus rel

Descrizione:
Configura timbro e ADSR di una voce.

- v = 1..4
- wave$ forma d’onda
- pw pulse width
- atk dec sus rel inviluppo

Esempio pratico – Confronto timbri

```
10 SIDINIT 16000
20 SIDVOL 5
30 SIDVOICE 2 "TRI" 0 2 6 10 4
40 SIDMCLR
50 SIDMADD 2 "A4" 400
60 SIDMPLAY 0
70 WAIT 500
80 SIDVOICE 2 "SAW" 0 1 4 12 2
90 SIDMPLAY 0
100 GOTO 100
```

Note

- "TRI" = più morbido (C64)
- "SAW" / "PULSE" = più aggressivi

SIDVOL

Sintassi:
SIDVOL vol

Descrizione:
Imposta il volume globale del motore audio.

Esempio pratico – Volume a confronto

```
10 SIDINIT 16000
20 SIDVOICE 2 "TRI" 0 2 6 10 4
30 SIDMCLR
40 SIDMADD 2 "A4" 500
50 SIDVOL 3
60 SIDMPLAY 0
70 WAIT 600
80 SIDVOL 7
90 SIDMPLAY 0
100 GOTO 100
```

Note

- Valori consigliati con PAM8403: **3..6**
- Volume alto senza attenuazione può distorcere.

SIN(x)

Sintassi:

SIN(x)

Descrizione:
La funzione SIN(x) restituisce il **seno** dell'angolo x, espresso in **radianti**.
Fa parte delle funzioni matematiche trigonometriche standard ed è utile in calcoli scientifici, grafica, simulazioni, ecc.

Per convertire gradi in radianti:

radianti = gradi * (PI / 180)

Esempi pratici

Esempio 1 – Seno di 0 radianti

```
10 PRINT SIN(0)
RUN
```

Output atteso:

```
0
```

Esempio 2 – Seno di PI/2 (~1.5708 radianti)

```
10 PRINT SIN(3.14159 / 2)
RUN
```

Output atteso:

```
1
```

Esempio 3 – Calcolo del seno da gradi
→ Angolo di 30°

```
10 G = 30
20 R = G * 3.14159 / 180
30 PRINT "SIN(30°) = "; SIN(R)
RUN
```

Output atteso:

```
SIN(30°) = 0.5
```

Esempio 4 – Tabella dei seni per angoli da 0 a 90°

```
10 FOR A = 0 TO 90 STEP 15
20 R = A * 3.14159 / 180
30 PRINT "SIN("; A; "°) = "; SIN(R)
40 NEXT A
RUN
```

Output atteso:

```
SIN(0°) = 0
SIN(15°) = 0.2588
SIN(30°) = 0.5
SIN(45°) = 0.7071
SIN(60°) = 0.8660
SIN(75°) = 0.9659
SIN(90°) = 1
```

Nota:

- Il risultato è compreso tra -1 e +1
- Per altre funzioni trigonometriche usa COS(x), TAN(x), ATN(x)

SPC(n)

Sintassi:

SPC(n)

Descrizione:
La funzione SPC(n) genera una **stringa di n spazi**, utile per **formattare l'output**, **allineare testi**, o **creare margini visivi** nel terminale.

Può essere utilizzata:

- All'interno di PRINT per creare spaziature
- Per costruire righe con colonne ben separate
- In combinazione con altre stringhe

Esempi pratici

Esempio 1 – Spazio tra due parole

```
10 PRINT "CIAO" + SPC(5) + "MONDO"
RUN
```

Output atteso:

```
CIAO     MONDO
```

Esempio 2 – Allineamento su più righe

```
10 PRINT "NOME" + SPC(10) + "ETA'"
20 PRINT "LUCA" + SPC(11) + "12"
30 PRINT "MARCO" + SPC(9) + "15"
RUN
```

Output atteso:

```
NOME         ETA'
LUCA         12
MARCO         15
```

Esempio 3 – Uso dinamico

```
10 FOR I = 1 TO 5
20 PRINT SPC(I) + "TEST"
30 NEXT I
RUN
```

Output atteso:

```
TEST
  TEST
   TEST
    TEST
     TEST
```

Esempio 4 – Rientro fisso

```
10 INDENT = 8
20 PRINT SPC(INDENT) + "RIGA FORMATTA"
RUN
```

Output atteso:

```
        RIGA FORMATTA
```

Nota:

- Se `n = 0`, non viene generato alcuno spazio
- Se `n` è maggiore della larghezza del terminale, il testo potrebbe andare a capo
- Combinabile con `TAB(n)` per posizionamenti più precisi

SPIFREE

Sintassi:
PRINT SPIFREE

Descrizione:
Il comando SPIFREE restituisce lo spazio libero disponibile nel file system SPIFFS (memoria interna).
Utile per verificare lo spazio residuo prima di salvare file o log.

Esempi pratici

Esempio 1 – Visualizzare spazio libero su SPIFFS

```
10 PRINT SPIFREE
RUN
```

Output atteso:

```
spazio disponibile
```

Nota:
• Il valore è espresso in byte
• Non richiede parametri
• Funziona solo se SPIFFS è inizializzato correttamente

SQR(x)

Sintassi:

SQR(x)

Descrizione:

La funzione **SQR(x)** restituisce la **radice quadrata** di un numero.
Accetta un valore numerico x e ritorna √x.

È utile per calcoli matematici, geometria, trigonometria, conversioni e in tutte le situazioni dove occorre estrarre una radice quadrata.

Condizione importante:
x deve essere **maggiore o uguale a 0**.
Se x < 0, viene generato un errore matematico.

Esempi pratici

Esempio 1 – Radice quadrata semplice

```
10 PRINT SQR(9)
RUN
```

Output atteso:

```
3
```

Esempio 2 – Uso con variabile

```
10 A = 49
20 PRINT SQR(A)
RUN
```

Output atteso:

```
7
```

Esempio 3 – Con funzione combinata

```
10 X = 16
20 Y = SQR(X) + 2
30 PRINT Y
RUN
```

Output atteso:

6

Esempio 4 – Errore su valori negativi

10 PRINT SQR(-4)
RUN

Output atteso:

RUNTIME ERROR: SQR domain error (x < 0)

Nota:

• x deve essere ≥ 0
• SQR(x) è calcolata internamente come sqrt(x)
• Ideale con TAN, SIN, COS, e per distanza Euclidea
• Puoi usarla anche dentro espressioni complesse

SR595 CLEAR

Sintassi:

SR595 CLEAR

Descrizione:
Imposta **tutte le uscite** della catena 74HC595 a LOW (0).
Equivalente a SR595 FILL 0.

Esempio pratico:

```
10 SR595 INIT 23 18 5
20 SR595 CLEAR
RUN
```

Note:

- Utile per **azzerare** rapidamente tutte le uscite.
- Se hai più chip, tutte le **8 × chips** uscite saranno portate a 0.

SR595 FILL

Sintassi:

SR595 FILL 0|1

Descrizione:
Imposta **tutte le uscite** a 0 (LOW) o 1 (HIGH).

Esempio pratico:

```
10 SR595 INIT 23 18 5
20 SR595 FILL 1   ' Tutte ON
30 WAIT 1000
40 SR595 FILL 0   ' Tutte OFF
RUN
```

Note:

- Funziona su **uno o più chip** contemporaneamente.
- Per testare velocemente LED/relè su tutte le linee.

SR595 INIT (SHIFT REGISTER 74HC595)

Sintassi:

SR595 INIT data clock latch [oe] [mr] [chips] [MSB|LSB]

Descrizione:
Inizializza uno o più registri **74HC595** e imposta i pin di controllo.

- data = pin DS (Dati in ingresso).
- clock = pin SHCP (Clock di shift).
- latch = pin STCP (Latch per aggiornare le uscite).
- oe (opz.) = pin OE (active-LOW; LOW = abilitato).
- mr (opz.) = pin MR (active-LOW; LOW = reset).
- chips (opz.) = numero di chip collegati in cascata (default 1).
- MSB|LSB (opz.) = ordine di invio dei bit (default MSB).

Esempi pratici:
Esempio 1 – Un solo chip:

```
10 SR595 INIT 23 18 5
```

Esempio 2 – Due chip in cascata, ordine LSB-first:

```
10 SR595 INIT 23 18 5 -1 -1 2 LSB
```

Collegamento con più chip (cascata):

- **DATA** (DS) del primo chip collegato al pin data del micro.
- **Q7'** del primo chip collegato al **DS** del secondo.
- **CLOCK (SHCP)**, **LATCH (STCP)**, **OE**, **MR** comuni a tutti.
- Imposta chips in INIT al numero totale di chip.

Consigli pratici per più 74HC595

- **Collegamento base:**
  - **MCU → DS (DATA) → Q7'→ DS chip successivo**
  - **MCU → SHCP (CLOCK) → tutti i chip**
  - **MCU → STCP (LATCH) → tutti i chip**
  - **OE (active-LOW)** e **MR (active-LOW)** comuni (OE=GND, MR=VCC se non usati).
- **Esempio 3 chip (24 uscite):**
- 10 SR595 INIT 23 18 5 -1 -1 3
- 20 SR595 WRITE &HFFFFFF ' tutte ON
- 30 WAIT 1000
- 40 SR595 CLEAR
- 50 FOR I=0 TO 23
- 60 SR595 SET I 1
- 70 WAIT 100
- 80 NEXT I
- 90 GOTO 50
- RUN

- **Driver esterni:** usa **ULN2803** o MOSFET per pilotare carichi come relè, motori, lampade.

Note:

- Dopo INIT, tutte le uscite sono **LOW**.
- Il buffer interno (SR595 WRITE, SR595 SET) gestisce **8 × chips** bit.

SR595 SET

Sintassi:

SR595 SET index 0|1

Descrizione:
Imposta **un singolo output** della catena a 0 o 1.

- index parte da **0** per Q0 del primo chip (vicino al micro),
- 7 = Q7 del primo chip,
- 8 = Q0 del secondo chip, e così via.

Esempio pratico:

```
10 SR595 INIT 23 18 5 -1 -1 2
20 SR595 SET 0 1    ' Q0 del primo chip ON
30 SR595 SET 8 1    ' Q0 del secondo chip ON
40 WAIT 1000
50 SR595 CLEAR
RUN
```

Note:

- Aggiorna immediatamente il latch interno.
- Utile per pilotare **relè singoli**, **LED specifici**, **linee di controllo**.

SR595 SHOW

Sintassi:

SR595 SHOW

Descrizione:
Riesegue il **latch** dello stato interno.
Forza l'uscita dell'ultimo stato caricato, anche senza modifiche.

Esempio pratico:

```
10 SR595 INIT 23 18 5
20 SR595 FILL 1
30 SR595 SHOW
RUN
```

Note:

- Serve solo in situazioni particolari (es. dopo reset hardware MR).
- SET, WRITE, FILL e CLEAR fanno già il latch automaticamente.

SR595 WRITE

Sintassi:

SR595 WRITE value

Descrizione:
Scrive un valore numerico su **tutte le uscite** (fino a 8 × chips bit).

- I bit **meno significativi** (LSB) corrispondono a Q0..Q7 del primo chip.
- I bit successivi ai chip successivi in cascata.

Esempio pratico:

```
10 SR595 INIT 23 18 5 -1 -1 2
20 SR595 WRITE &H00FF   ' Chip1=FF (tutti ON), Chip2=00 (tutti OFF)
30 WAIT 1000
40 SR595 WRITE &HFF00   ' Chip1=OFF, Chip2=ON
RUN
```

Note:

- Sovrascrive **tutto lo stato**: per modifiche parziali usa SET.
- Per più chip, considera la rappresentazione in binario/esadecimale per comodità.

STARTFUNC / STOPFUNC

Sintassi:
STARTFUNC <nome>
STOPFUNC <nome>

Descrizione:
STARTFUNC avvia in **modalità non bloccante** una funzione definita con FUNC <nome> LOOP, che continuerà a girare in background.
STOPFUNC interrompe l'esecuzione continua della funzione indicata.

Esempi pratici

Esempio 1 – Funzione eseguita una sola volta (CALLFUNC)

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC FLASH
20 DWRITE 2 1
30 DELAY 300
40 DWRITE 2 0
50 DELAY 300
60 ENDFUNC
70 CALLFUNC FLASH
RUN
```

Output atteso:
Il LED sul pin 2 lampeggia una volta (accende per 300 ms, poi spegne).

Esempio 2 – Funzione ciclica in background (FUNC ... LOOP + STARTFUNC)

```
5 PINMODE 2 OUTPUT NOPULL
10 FUNC BLINK LOOP
20 DWRITE 2 1
30 DELAY 500
40 DWRITE 2 0
50 DELAY 500
60 ENDFUNC
70 STARTFUNC BLINK
RUN
```

Output atteso:
Il LED lampeggia continuamente ogni 500 ms, senza bloccare il resto del programma.

Esempio 3 – Fermare una funzione ciclica

```
10 STOPFUNC BLINK
RUN
```

Output atteso:
Il LED smette di lampeggiare.

Nota:

• Il nome della funzione deve essere unico e senza spazi
• Le funzioni **normali** si richiamano con CALLFUNC
• Le funzioni con LOOP si eseguono in parallelo con STARTFUNC e si fermano con STOPFUNC
• Ogni FUNC deve sempre essere chiusa con ENDFUNC
• All'interno della funzione si possono usare comandi BASIC standard (PRINT, IF, WAIT, DWRITE, ecc.)
• Può essere usato per multitasking (es. sensori, output, controlli periodici)

STOP, CONT, END

Sintassi:

STOP
CONT
END

Descrizione:

STOP

Sospende l'esecuzione del programma in **modo volontario**.
L'utente può poi usare CONT per riprendere l'esecuzione **dal punto in cui era stata fermata**.

Utile per debug o per mettere in pausa l'esecuzione.

CONT

Riprende l'esecuzione **dal punto in cui è stato eseguito un STOP**.
Non funziona se il programma è stato interrotto con END, RUN, oppure dopo un errore.

Se usato senza un precedente STOP, non ha effetto.

END

Termina **del tutto** il programma in corso.
Dopo l'END, non è possibile usare CONT.
L'END è opzionale: se non presente, il programma termina automaticamente all'ultima riga.

Esempi pratici

Esempio 1 – Uso di STOP e CONT

```
10 INPUT "INSERISCI UN NUMERO: "; N
20 IF N = 0 THEN STOP
30 PRINT "NUMERO VALIDO: "; N
RUN
```

Output atteso:

```
INSERISCI UN NUMERO: ? 0
(STOP)
```

Poi:

```
CONT
```

Output:
Riprende l'esecuzione dalla riga successiva al STOP.

Esempio 2 – Uso di END

```
10 PRINT "INIZIO"
20 END
30 PRINT "QUESTO NON VIENE MAI ESEGUITO"
```

Output atteso:

```
INIZIO
```

Esempio 3 – STOP condizionato + CONT manuale

```
10 FOR I = 1 TO 5
20 PRINT "PASSO"; I
30 IF I = 3 THEN STOP
40 NEXT I
```

Output dopo RUN:

```
PASSO 1
PASSO 2
PASSO 3
(STOP)
```

Dopo:

```
CONT
```

Output finale:

```
PASSO 4
PASSO 5
```

Note:

- STOP è utile per debug o pause logiche
- CONT funziona **solo dopo STOP**, **non dopo END o RUN**
- END chiude il programma in modo pulito

STR$(x) o STR$(x, n)

Sintassi:

STR$(numero)
STR$(numero, decimali da visualizzare)

Descrizione:

La funzione STR$ converte un **numero** (intero o decimale) in **stringa di testo**.
È utile quando si vuole **concatenare numeri a stringhe**, oppure **salvare/stampare valori** in formato testuale.

Il numero convertito **mantiene il segno** e viene trasformato in una stringa compatibile con altre funzioni stringa (es. LEFT$, RIGHT$, LEN).

Esempi pratici

Esempio 1 – Conversione base

```
10 X = 123
20 PRINT STR$(X)
RUN
```

Output atteso:

```
123
```

Esempio 2 – Concatenazione con testo

```
10 V = 42
20 PRINT "IL VALORE È " + STR$(V)
RUN
```

Output atteso:

```
IL VALORE È 42
```

Esempio 3 – Numeri negativi

```
10 A = -17
20 PRINT STR$(A)
RUN
```

Output atteso:

```
-17
```

Esempio 4 – Uso in salvataggio dati

```
10 T = 88
20 RIGA$ = "TOTALE=" + STR$(T)
30 SAVEINT "totale.txt"
RUN
```

(Supponendo che venga salvato il contenuto del listato con valore convertito in testo)

Esempio 5 – Uso combinato con LEN

```
10 N = 12345
20 S$ = STR$(N)
30 PRINT "CIFRE: "; LEN(S$)
RUN
```

Output atteso:

```
CIFRE: 5
```

Nota:

- Il risultato di STR$ è una **stringa numerica**, che può essere convertita nuovamente in numero con VAL(A$)
- Compatibile con tutti gli operatori stringa e con INPUT, PRINT, SAVE, ecc.

SYNCNTP

Sintassi:
SYNCNTP

Descrizione:
Il comando **SYNCNTP** sincronizza l'orologio interno del sistema con un server NTP (Network Time Protocol) tramite connessione Wi-Fi.
Una volta eseguito, l'orario di sistema viene aggiornato automaticamente con data e ora correnti ottenute via internet.

Esempi pratici

Esempio 1 – Sincronizzazione dell'orologio

```
10 SYNCNTP
RUN
```

Output atteso:
Viene sincronizzata la data e l'ora esatta aggiornate via rete.

Nota:
• È richiesta una connessione Wi-Fi attiva al momento dell'esecuzione
• Il fuso orario predefinito è UTC, ma può essere modificato con il comando TIMEZONE
• Non richiede parametri
• Utile per applicazioni che necessitano di orario esatto (es. data logging, schedulazioni)

TAN(x)

Sintassi:

TAN(x)

Descrizione:
La funzione TAN(x) restituisce la **tangente** dell'angolo x, espresso in **radianti**.
È calcolata come:

TAN(x) = SIN(x) / COS(x)

La tangente ha **valori compresi tra -∞ e +∞**, con **discontinuità** in corrispondenza di angoli per cui COS(x) = 0 (come π/2, 3π/2, ecc.).

Esempi pratici

Esempio 1 – Tangente di 0 radianti

```
10 PRINT TAN(0)
RUN
```

Output atteso:

```
0
```

Esempio 2 – Tangente di 45 gradi (π/4 radianti)

```
10 PI = 3.14159
20 PRINT TAN(PI / 4)
RUN
```

Output atteso:

```
1
```

Esempio 3 – Tangente di 60 gradi
→ Calcolo da gradi a radianti

```
10 A = 60
20 R = A * 3.14159 / 180
30 PRINT "TAN("; A; "°) = "; TAN(R)
RUN
```

Output atteso:

```
TAN(60°) = 1.732
```

Esempio 4 – Valori critici (attenzione alla discontinuità)
→ A 90° la tangente tende all'infinito

```
10 PI = 3.14159
20 PRINT TAN(PI / 2)
RUN
```

Output atteso:

(grande valore o errore numerico)

Nota:

- L’argomento x deve essere in **radianti**
- Per evitare errori, evitare angoli in cui COS(x) = 0
- Usare con SIN(x) e COS(x) per rappresentazioni geometriche

TCSFILTER

Sintassi:

TCSFILTER RED
TCSFILTER GREEN
TCSFILTER BLUE
TCSFILTER CLEAR

Descrizione:
Il comando TCSFILTER seleziona manualmente quale filtro colore attivare sul sensore:

- RED → filtro rosso
- GREEN → filtro verde
- BLUE → filtro blu
- CLEAR → nessun filtro, luce totale

Esempi pratici

Esempio 1 – Impostare filtro rosso

10 TCSINIT 17 16 4 2 15 5
20 TCSFILTER RED
30 PRINT "Filtro ROSSO attivo"
RUN

Output atteso:

Filtro ROSSO attivo

Esempio 2 – Impostare filtro trasparente

10 TCSINIT 17 16 4 2 15 5
20 TCSFILTER CLEAR
30 PRINT "Filtro CHIARO attivo"
RUN

Output atteso:

Filtro CHIARO attivo

Note:

- Normalmente non serve usarlo, perché i comandi TCSREAD e TCSRGB cambiano automaticamente i filtri.
- Può essere utile per **debug** o per misurare solo un canale alla volta.

TCSINIT (Sensore colore TCS230/TCS3200)

Sintassi:

TCSINIT S0 S1 S2 S3 OUT [LEDPIN]

Descrizione:
Il comando TCSINIT inizializza il sensore assegnando i pin usati:

- S0, S1, S2, S3 → pin digitali per la configurazione del sensore
- OUT → pin di ingresso che legge la frequenza dal sensore
- LEDPIN (opzionale) → pin che controlla l'illuminazione LED del modulo

---

Esempi pratici

Esempio 1 – Inizializzazione completa

```
10 TCSINIT 17 16 4 2 15 5
20 PRINT "Sensore inizializzato con LED"
RUN
```

Output atteso:

```
Sensore inizializzato con LED
```

Esempio 2 – Inizializzazione senza LED

```
10 TCSINIT 17 16 4 2 15
20 PRINT "Sensore inizializzato senza LED"
RUN
```

Output atteso:

```
Sensore inizializzato senza LED
```

Note:

- Deve essere il **primo comando** del sensore nel programma.
- Se non usi il LED, il parametro LEDPIN può essere omesso.
- Dopo TCSINIT, la scala predefinita è al 20%.

TCSLED

Sintassi:

TCSLED ON
TCSLED OFF

Descrizione:
Il comando TCSLED controlla l'illuminazione LED del modulo:

- ON → accende il LED bianco integrato
- OFF → spegne il LED

Esempi pratici

Esempio 1 – Accendere il LED

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 PRINT "LED acceso"
RUN
```

Output atteso:

LED acceso

Esempio 2 – Blink del LED

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 DELAY 500
40 TCSLED OFF
50 DELAY 500
60 TCSLED ON
RUN
```

Output atteso:
Il LED lampeggia due volte.

Note:

- Funziona solo se in TCSINIT hai specificato un pin LEDPIN.
- Utile per avere condizioni di luce costanti durante la misura.

TCSREAD

Sintassi:

TCSREAD varR varG varB varC

Descrizione:
Il comando TCSREAD esegue la lettura completa dei 4 canali:

- varR → valore Rosso
- varG → valore Verde
- varB → valore Blu
- varC → valore Clear (luce totale)

I valori restituiti sono frequenze in Hz.

Esempi pratici

Esempio 1 – Lettura singola

```
10 TCSINIT 17 16 4 2 15 5
20 TCSLED ON
30 TCSSCALE 20
40 TCSREAD R G B C
50 PRINT "R=";R;" G=";G;" B=";B;" C=";C
RUN
```

Output atteso (valori variabili):

```
R=1450  G=1780  B=920  C=4200
```

Esempio 2 – Lettura ripetuta in loop

```
10 TCSINIT 17 16 4 2 15 5
20 FOR I=1 TO 3
30   TCSREAD ROSSO VERDE BLU CHIARO
40   PRINT I;") ";ROSSO;",";VERDE;",";BLU;",";CHIARO
50   DELAY 500
60 NEXT I
RUN
```

Output atteso:
Tre righe di valori RGB+C diversi per ogni iterazione.

Note:

- Le variabili (R, G, B, C, ecc.) sono a scelta dell’utente.
- Se i valori sono troppo alti o bassi, usare TCSSCALE per regolare la sensibilità.
- È il comando più usato per analisi del colore.

TCSRGB

Sintassi:

TCSRGB varR varG varB

Descrizione:
Il comando TCSRGB legge solo i tre canali principali (Rosso, Verde e Blu), ignorando il canale Clear.

Esempi pratici

Esempio 1 – Lettura RGB

```
10 TCSINIT 17 16 4 2 15 5
20 TCSRGB R G B
30 PRINT "RGB: ";R;",";G;",";B
RUN
```

Output atteso (valori variabili):

RGB: 1500,1800,950

Esempio 2 – Identificazione colore dominante

```
10 TCSINIT 17 16 4 2 15 5
20 TCSRGB R G B
30 IF R>G AND R>B THEN PRINT "COLORE ROSSO"
40 IF G>R AND G>B THEN PRINT "COLORE VERDE"
50 IF B>R AND B>G THEN PRINT "COLORE BLU"
RUN
```

Output atteso (dipende dal campione):

COLORE ROSSO

Note:

- Più veloce di TCSREAD perché ignora il canale Clear.
- Utile per riconoscere il colore dominante di un oggetto.

TCSSCALE

Sintassi:

TCSSCALE n

Descrizione:
Il comando TCSSCALE imposta la scala di frequenza di uscita del sensore:

- 0 → Power Down (sensore spento)
- 2 → Scala 2% (per luci molto intense)
- 20 → Scala 20% (default)
- 100 → Scala 100% (per luci deboli)

Esempi pratici

Esempio 1 – Scala al 2%

```
10 TCSINIT 17 16 4 2 15 5
20 TCSSCALE 2
30 PRINT "Scala impostata al 2%"
RUN
```

Output atteso:

Scala impostata al 2%

Esempio 2 – Spegnere il sensore

```
10 TCSINIT 17 16 4 2 15 5
20 TCSSCALE 0
30 PRINT "Sensore in standby"
RUN
```

Output atteso:

Sensore in standby

Note:

- Usare valori bassi (2 o 20) in ambienti luminosi per evitare saturazioni.
- Usare 100 in ambienti bui per aumentare la sensibilità.
- Se il sensore deve essere temporaneamente disattivato, impostare 0.

TI

Sintassi:

TI

Descrizione:

TI è una **variabile di sistema** che rappresenta il **tempo trascorso** dall'accensione o dal reset dell'ESP32, espresso in **centesimi di secondo** (1 unità = 10 ms).
È **solo in lettura** e viene utilizzata per misurare intervalli di tempo, ritardi, o eventi temporizzati.

È particolarmente utile in combinazione con:

- DELAYMILLIS (per verificare il tempo passato)
- WHILE/WEND per pause non bloccanti
- IF per condizioni basate su durata

Esempi pratici

Esempio 1 – Stampa del timer corrente

```
10 PRINT "TI = "; TI
RUN
```

Output atteso:

```
TI = 1275
```

(Valore esemplificativo: 12,75 secondi dal boot)

Esempio 2 – Misura del tempo tra due punti

```
10 T0 = TI
20 FOR I = 1 TO 1000
30 NEXT I
40 DT = TI - T0
50 PRINT "TEMPO TRASCORSO = "; DT; " (centesimi di secondo)"
RUN
```

Output atteso:

```
TEMPO TRASCORSO = 15
```

Esempio 3 – Timer non bloccante con WHILE

```
10 T = TI
20 PRINT "Attendo 3 secondi..."
30 WHILE TI < T + 300: WEND
40 PRINT "Fatto!"
RUN
```

Output atteso:

```
Attendo 3 secondi...
(Fermo per 3 secondi)
Fatto!
```

Nota:

- TI si **azzererà** se il dispositivo viene riavviato
- È espresso in **centesimi di secondo**: moltiplica per 10 per ottenere millisecondi, dividi per 100 per ottenere secondi
- Per ottenere l'orario formattato, usa TI$

TI$

Sintassi:

TI$

Descrizione:

TI$ è una **variabile di sistema** in formato **stringa** che restituisce il tempo trascorso dall'accensione dell'ESP32 nel formato **hhmmss** (ore, minuti, secondi).
È utile per:

- Stampare il tempo in formato leggibile
- Registrare l'ora di un evento
- Mostrare orari o durate in forma compatta

L'orologio parte da 000000 all'avvio del dispositivo o dopo un REBOOT.

Esempi pratici

Esempio 1 – Visualizzare l'ora corrente

```
10 PRINT "TEMPO ATTUALE: "; TI$
RUN
```

Output atteso:

```
TEMPO ATTUALE: 000315
```

(Esempio: 3 minuti e 15 secondi dal boot)

Esempio 2 – Mostrare tempo durante un programma

```
10 PRINT "INIZIO ALLE: "; TI$
20 DELAY 2000
30 PRINT "FINE ALLE: "; TI$
RUN
```

Output atteso:

```
INIZIO ALLE: 000000
FINE ALLE: 000002
```

Esempio 3 – Scrivere in un file con timestamp

```
10 T$ = "LOG " + TI$ + ": PROGRAMMA AVVIATO"
20 PRINT T$
30 SAVEINT "log.txt"
```

```
RUN
```

Output atteso:

```
LOG 000120: PROGRAMMA AVVIATO
```

Esempio 4 – Verifica se tempo supera 10 minuti

```
10 H = VAL(LEFT$(TI$, 2))
20 M = VAL(MID$(TI$, 3, 2))
30 IF H = 0 AND M >= 10 THEN PRINT "OLTRE 10 MINUTI"
RUN
```

Output atteso:
Se sono passati più di 10 minuti, verrà stampato il messaggio.

Nota:

- Il formato è **stringa esattamente di 6 cifre**
- Può essere analizzata con LEFT$, MID$, VAL per ottenere ore, minuti, secondi separatamente
- Si aggiorna automaticamente con il passare del tempo (come TI, ma formattato)

TIMEH

Sintassi:
TIMEH

Descrizione:
Il comando **TIMEH** restituisce l'**ora** corrente (0–23) secondo l'orologio interno del sistema. È utile per controlli basati sull'orario (es. automazioni orarie).

Esempi pratici

Esempio 1 – Stampare l'ora corrente

```
10 PRINT TIMEH
RUN
```

Output atteso:

```
14
```

Nota:
• Restituisce un intero da 0 a 23
• Non richiede parametri
• Utile in controlli di precisione temporale

TIMEM

Sintassi:
TIMEM

Descrizione:
Il comando **TIMEM** restituisce i **minuti** correnti (0–59) secondo l'orologio interno.

Esempi pratici

Esempio 1 – Stampare i minuti correnti

```
10 PRINT TIMEM
RUN
```

Output atteso:

```
38
```

Nota:
• Restituisce un intero da 0 a 59
• Non richiede parametri
• Utile in controlli di precisione temporale

TIMES

Sintassi:
TIMES

Descrizione:
Il comando **TIMES** restituisce i **secondi** correnti (0–59) dell'orologio interno.
Permette controlli al secondo o temporizzazioni di breve durata.

Esempi pratici

Esempio 1 – Stampare i secondi correnti

```
10 PRINT TIMES
RUN
```

Output atteso:

```
05
```

Nota:
• Restituisce un intero da 0 a 59
• Non richiede parametri
• Utile in controlli di precisione temporale

TIMEZONE

Sintassi

TIMEZONE codice
TIMEZONE "regola_POSIX"
TIMEZONE +H
TIMEZONE -H
TIMEZONE +H:MM
TIMEZONE -H:MM
TIMEZONE HELP
TIMEZONE ?

Descrizione

TIMEZONE imposta il fuso orario (timezone) usato dal sistema per calcolare l'ora locale.

- codice è un identificatore breve (IT, US, JP, …) che viene tradotto automaticamente in una stringa di fuso orario in formato POSIX.
- "regola_POSIX" permette di impostare direttamente una regola POSIX completa (inclusa l'ora legale se presente).
- +H, -H, +H:MM, -H:MM consentono di impostare manualmente l'offset rispetto a UTC in modo rapido:
    - +1 significa UTC+1
    - -5 significa UTC-5
    - +5:30 significa UTC+5 ore e 30 minuti
      Questo viene convertito internamente nel formato POSIX corretto (dove il segno è invertito).

L'impostazione del fuso orario viene applicata immediatamente e viene anche riutilizzata dal comando SYNCNTP, quindi resta valida dopo la sincronizzazione NTP.

Non viene cambiato l'orologio hardware interno: cambia solo **come** vengono interpretati data e ora locali rispetto a UTC.

Argomenti speciali

- TIMEZONE HELP
- TIMEZONE ?

Mostrano l'elenco completo dei codici supportati e le rispettive regole POSIX.
Questa funzione sostituisce il vecchio comando LISTTIMEZONES.

Codici disponibili

Questi codici sono già mappati internamente:

Codice	Regola POSIX	Area geografica / Note
IT	CET-1CEST,M3.5.0/2,M10.5.0/3	Italia
FR	CET-1CEST,M3.5.0/2,M10.5.0/3	Francia
DE	CET-1CEST,M3.5.0/2,M10.5.0/3	Germania
UK	GMT0BST,M3.5.0/1,M10.5.0/2	Regno Unito
ES	CET-1CEST,M3.5.0/2,M10.5.0/3	Spagna
US	EST5EDT,M3.2.0/2,M11.1.0/2	Stati Uniti (costa Est, es. New York)
CA	EST5EDT,M3.2.0/2,M11.1.0/2	Canada (zona Est)
JP	JST-9	Giappone (nessuna ora legale)
CN	CST-8	Cina (nessuna ora legale)
IN	IST-5:30	India (offset frazionato, nessuna ora legale)
AU	AEST-10AEDT,M10.1.0/2,M4.1.0/3	Australia (Sydney)
BR	BRT3BRST,M10.3.0/0,M2.3.0/0	Brasile
RU	MSK-3	Russia (Mosca)
MX	CST6CDT,M4.1.0,M10.5.0	Messico
AR	ART3	Argentina
ZA	SAST-2	Sudafrica
EG	EET-2	Egitto
KR	KST-9	Corea del Sud (nessuna ora legale)
SE	CET-1CEST,M3.5.0/2,M10.5.0/3	Svezia
CH	CET-1CEST,M3.5.0/2,M10.5.0/3	Svizzera
NL	CET-1CEST,M3.5.0/2,M10.5.0/3	Paesi Bassi
BE	CET-1CEST,M3.5.0/2,M10.5.0/3	Belgio
PL	CET-1CEST,M3.5.0/2,M10.5.0/3	Polonia
GR	EET-2EEST,M3.5.0/3,M10.5.0/4	Grecia
FI	EET-2EEST,M3.5.0/3,M10.5.0/4	Finlandia
NO	CET-1CEST,M3.5.0/2,M10.5.0/3	Norvegia
PT	WET0WEST,M3.5.0/1,M10.5.0/2	Portogallo
DK	CET-1CEST,M3.5.0/2,M10.5.0/3	Danimarca
IE	GMT0IST,M3.5.0/1,M10.5.0/2	Irlanda
UTC	UTC0	Tempo Coordinato Universale (nessun offset)

Nota: le regole POSIX includono anche il passaggio automatico ora solare / ora legale dove previsto (es. CET-1CEST,...).

Uso con offset manuale

Puoi impostare un fuso orario personalizzato senza usare un codice esistente.

Esempi validi:

TIMEZONE +1
TIMEZONE -5
TIMEZONE +5:30

Significato:

- TIMEZONE +1 imposta l'ora locale a UTC+1.
- TIMEZONE -5 imposta l'ora locale a UTC-5.
- TIMEZONE +5:30 imposta l'ora locale a UTC+5 ore e 30 minuti.

Internamente, questi valori vengono convertiti automaticamente in una regola POSIX equivalente (ad esempio UTC-1 per UTC+1), così il sistema li può usare subito.

Questa modalità non definisce l'ora legale: è un offset fisso.

Esempi pratici

Esempio 1 – Impostare il fuso orario italiano e sincronizzare

10 TIMEZONE IT
20 SYNCNTP
RUN

Output atteso: data e ora correnti in formato locale italiano, con gestione automatica ora solare / ora legale.

Esempio 2 – Impostare manualmente UTC+1 senza tabella

TIMEZONE +1
SYNCNTP

Esempio 3 – Visualizzare la lista dei fusi orari supportati

TIMEZONE HELP

oppure

TIMEZONE ?

Output atteso: elenco codici (IT, US, JP, …) con la relativa definizione POSIX.

Note finali

- TIMEZONE HELP / TIMEZONE ? ha sostituito LISTTIMEZONES.
- Il cambiamento della timezone è immediato.
- L’orario interno (RTC/contatore di sistema) rimane in UTC; cambia solo la conversione a ora locale.
- Dopo aver scelto il fuso orario corretto, l’uso di SYNCNTP aggiorna l’orologio in rete e ti dà data/ora locale giusta.

TONE

Sintassi:

TONE pin freq durata

Descrizione:
Emette un suono sul pin specificato, con una frequenza espressa in Hertz (freq) o come nome di nota tra virgolette (es. "A4"), per una durata specificata in millisecondi (durata).

Esempi pratici

10 TONE 26 440 500

Emette un suono a 440 Hz (nota **LA4**) per 500 ms sul pin 26.

20 TONE 26 "C5" 300

Emette un DO della quinta ottava per 300 ms.

Note

- Il pin deve essere collegato a un buzzer piezoelettrico o un altoparlante.
- Se si usa una **nota tra virgolette**, il sistema convertirà automaticamente la nota nella relativa frequenza.
- Le virgolette devono essere **doppie** (") e **obbligatorie** per indicare una nota (es. "A4"), non vanno usate per frequenze numeriche.

Frequenze comuni

Nota	Frequenza (Hz)
DO	262
RE	294
MI	330
FA	349
SOL	392
LA	440
SI	494

Mappa note disponibili

(da **C1** a **B7**)

Ottava	Note (frequenza in Hz)
C1–B1	C1 33, C#1 35, D1 37, D#1 39, E1 41, F1 44, F#1 46, G1 49, G#1 52, A1 55, A#1 58, B1 62
C2–B2	C2 65, C#2 69, D2 73, D#2 78, E2 82, F2 87, F#2 92, G2 98, G#2 104, A2 110, A#2 117, B2 123
C3–B3	C3 131, C#3 139, D3 147, D#3 156, E3 165, F3 175, F#3 185, G3 196, G#3 208, A3 220, A#3 233, B3 247
C4–B4	C4 262, C#4 277, D4 294, D#4 311, E4 330, F4 349, F#4 370, G4 392, G#4 415, A4 440, A#4 466, B4 494
C5–B5	C5 523, C#5 554, D5 587, D#5 622, E5 659, F5 698, F#5 740, G5 784, G#5 831, A5 880, A#5 932, B5 988
C6–B6	C6 1047, C#6 1109, D6 1175, D#6 1245, E6 1319, F6 1397, F#6 1480, G6 1568, G#6 1661, A6 1760, A#6 1865, B6 1976
C7–B7	C7 2093, C#7 2217, D7 2349, D#7 2489, E7 2637, F7 2794, F#7 2960, G7 3136, G#7 3322, A7 3520, A#7 3729, B7 3951

ULTRA INIT (SENSORE ULTRASUONI HC-SR04)

Sintassi:
ULTRA INIT <trig> <echo> [timeout_us]

Descrizione:
Configura i pin TRIG ed ECHO del sensore HC-SR04. Il parametro opzionale imposta il timeout di lettura in microsecondi.
Non produce output; abilita le letture successive con ULTRA READ.

Esempi pratici

Esempio 1 – Inizializzazione base
10 ULTRA INIT 5 18
20 ULTRA READ
RUN
Output atteso:
un numero in cm (es. 42.15)

Esempio 2 – Timeout personalizzato
10 ULTRA INIT 5 18 40000
20 ULTRA READ
RUN
Output atteso:
un numero in cm (es. 57.82)

Nota:
• Usare ULTRA INIT prima di ULTRA READ
• Timeout di default ≈ 30000 µs (~5 m)
• Il pin ECHO dell'HC-SR04 è a 5 V: usare un partitore verso l'ESP32; TRIG a 3.3 V va bene

ULTRA READ

Sintassi:
ULTRA READ [samples] [var]

Descrizione:
Misura la distanza in centimetri.
Senza var stampa direttamente la distanza (in cm). Con var assegna il valore a una variabile (numerica o stringa se termina con $).
Il parametro facoltativo samples indica il numero di campioni da mediare per una lettura più stabile.

Esempi pratici

Esempio 1 – Stampa diretta (media 3)
10 ULTRA INIT 5 18
20 ULTRA READ 3
RUN
Output atteso:
un numero in cm per riga (es. 41.97)

Esempio 2 – Assegnazione a variabile e soglia
10 ULTRA INIT 5 18
20 ULTRA READ 5 D
30 PRINT D
40 IF D < 20 THEN PRINT "TROPPO VICINO"
RUN
Output atteso:
18.34
TROPPO VICINO

Esempio 3 – Assegnazione a stringa
10 ULTRA INIT 5 18
20 ULTRA READ 3 S$
30 PRINT S$
RUN
Output atteso:
es. 33.20

Nota:
• Funziona dopo ULTRA INIT
• In caso di timeout: stampa “NaN” o assegna NaN alla variabile numerica
• samples consigliato 3–7 per stabilità
• Compatibile con PRINT, LET, IF (usando una variabile)

ULTRA TIMEOUT

Sintassi:
ULTRA TIMEOUT <timeout_us>

Descrizione:
Imposta il timeout globale (in microsecondi) per le letture ultrasoniche.
Non stampa nulla; il nuovo valore si applica alle letture successive di ULTRA READ.

Esempi pratici

Esempio 1 – Aumentare la portata
10 ULTRA INIT 5 18
20 ULTRA TIMEOUT 40000
30 ULTRA READ
RUN
Output atteso:
un numero in cm

Esempio 2 – Ridurre la latenza
10 ULTRA INIT 5 18
20 ULTRA TIMEOUT 20000
30 ULTRA READ 3 D
40 PRINT D
RUN
Output atteso:
cm mediati (es. 35.44)

Nota:
• Valore richiesto > 0
• Da usare dopo ULTRA INIT; influenza tutte le letture successive
• Scegliere timeout maggiore per distanze più lunghe, minore per risposta più rapida

VAL(A$)

Sintassi:

VAL(stringa$)

Descrizione:
La funzione VAL converte una **stringa numerica** in un **valore numerico** (intero o con decimali).
È utile per:

- Interpretare input testuali come numeri
- Eseguire calcoli su dati ricevuti come stringhe
- Leggere valori numerici salvati in file o da input seriale

Se la stringa non inizia con un numero valido, il risultato sarà 0.

Esempi pratici

Esempio 1 – Conversione semplice

```
10 S$ = "123"
20 N = VAL(S$)
30 PRINT N + 1
RUN
```

Output atteso:

```
124
```

Esempio 2 – Uso diretto con INPUT

```
10 INPUT "INSERISCI UN NUMERO COME STRINGA: "; T$
20 V = VAL(T$)
30 PRINT "NUMERO DOPPIO: "; V * 2
RUN
```

Input esempio:

```
INSERISCI UN NUMERO COME STRINGA: ? "50"
```

Output atteso:

```
NUMERO DOPPIO: 100
```

Esempio 3 – Conversione di decimali

```
10 S$ = "3.14"
```

```
20 PRINT VAL(S$) * 2
RUN
```

Output atteso:

```
6.28
```

Esempio 4 – Parte iniziale non numerica

```
10 A$ = "ABC123"
20 PRINT VAL(A$)
RUN
```

Output atteso:

```
0
```

Esempio 5 – Confronto con `STR$`

```
10 X = 77
20 S$ = STR$(X)
30 Y = VAL(S$)
40 PRINT Y + 1
RUN
```

Output atteso:

```
CopiaModifica
78
```

Nota:

- Legge solo il **numero iniziale** nella stringa
- Se la stringa è vuota o non numerica, restituisce `0`
- Inverso logico di `STR$`

VERIFY

(o **VERIFY F$**)

Sintassi:

VERIFY "nomefile"
VERIFY variabile$
VERIFY SD "nomefile"
VERIFY SD variabile$
VERIFY SPIFFS "nomefile"
VERIFY SPIFFS variabile$

Descrizione:

VERIFY confronta il programma attualmente in memoria con il contenuto del file indicato.

- Senza specificare la memoria, verifica in **SPIFFS** (memoria interna).
- Con SD, verifica il file sulla **scheda SD**.
- È possibile indicare esplicitamente SPIFFS per chiarezza.
- Il nome del file può essere passato **tra virgolette** oppure tramite **variabile stringa** (es. F$).

Esito possibile:

- File **uguale** al listato in memoria → corrisponde
- File **diverso** dal listato in memoria → differente
- **File non trovato** → errore

L'operazione è **solo di controllo**: non modifica nulla.

Esempi pratici

Esempio 1 – Verificare un file in SPIFFS (default)

VERIFY "main.bas"

Output atteso (se identico):

VERIFY successful (SPIFFS): content matches.

Esempio 2 – Verificare un file su SD

VERIFY SD "backup.bas"

Output atteso (se differente):

```
VERIFY FAILED: line mismatch.
EXPECTED: 10 PRINT "CIAO"
FOUND:    10 PRINT "HELLO"
```

Esempio 3 – Usare una variabile stringa

```
10 LET F$ = "setup.bas"
20 VERIFY F$
RUN
```

Nota:

- Se il file contiene **righe extra** rispetto al listato in memoria, viene segnalato come differente.
- Se la scheda SD non è presente/montata, VERIFY SD ... restituirà un errore di lettura.
- Consigliato usare l’estensione **.bas** per coerenza con gli altri comandi.

VREAD(p)

Sintassi:

VREAD(p)
VREAD(p) <R1> <R2>

Descrizione:

VREAD restituisce la **tensione in Volt**:

- Con una sola forma: misura la **tensione presente al pin** (Vpin).
- Con partitore: ricostruisce la **tensione a monte** di un partitore resistivo R1–R2 collegato al pin (nodo centrale). Formula:
  V_in = V_pin * (R1 + R2) / R2

Esempi pratici

Esempio 1 – Lettura diretta sul pin 34

```
10 V = VREAD(34)
20 PRINT "Vpin=", V
```

⟶ Mostra la tensione misurata al nodo del pin.

Esempio 2 – Lettura batteria con partitore (100k/100k) su pin 34

```
10 V = VREAD(34) 100000 100000
20 PRINT "VBAT=", V
```

⟶ Ricostruisce la tensione della batteria (≈ 2 × Vpin).

Esempio 3 – Con variabili

```
10 PIN=34: R1=100000: R2=100000
20 VB = VREAD(PIN) R1 R2
30 PRINT "VBAT=", VB
```

Output atteso:

```
Vpin= 0.012345
VBAT= 3.274910
```

Nota:

- VREAD usa i parametri di **ADC CAL** (REF/GAIN/OFFSET).
- Con il partitore, R2 **deve** essere >0 (altrimenti errore).
- Se il pin è flottante potresti leggere valori casuali.
- Non superare 3.3 V sul pin ADC dell'ESP32.

WAIT n

Sintassi:

WAIT n

Descrizione:
Il comando WAIT sospende l'esecuzione del programma per n **millisecondi**.
Durante il ritardo, il microcontrollore **non esegue altro codice**: è una pausa **bloccante**.

È utile per:

- Attendere tra due operazioni
- Creare animazioni o lampeggi
- Simulare tempi di caricamento

Esempi pratici

Esempio 1 – Pausa tra due messaggi

```
10 PRINT "CIAO"
20 WAIT 1000
30 PRINT "MONDO"
RUN
```

Output atteso (con 1 secondo di pausa tra le due righe):

```
CIAO
(monitora pausa)
MONDO
```

Esempio 2 – Lampeggio simulato

```
10 CLS
20 PRINT "☼"
30 WAIT 500
40 CLS
50 WAIT 500
60 GOTO 10
RUN
```

Output atteso:
Un lampeggio infinito del simbolo "☼" ogni mezzo secondo.

Esempio 3 – Ritardo dopo lettura

```
10 INPUT "INSERISCI IL TUO NOME: "; N$
20 PRINT "ATTENDI..."
```

```
30 WAIT 2000
40 PRINT "BENVENUTO "; N$
RUN
```

Output atteso:
Dopo l’input, una pausa di 2 secondi prima del messaggio di benvenuto.

Esempio 4 – Conto alla rovescia

```
10 FOR I = 5 TO 1 STEP -1
20 PRINT I
30 WAIT 1000
40 NEXT I
50 PRINT "VIA!"
RUN
```

Output atteso:
Un countdown da 5 a 1 con 1 secondo tra ciascun numero.

Nota:

- WAIT blocca **totalmente** l’esecuzione (incluso INPUT, GET, ecc.)
- L'unità di misura è il **millisecondo** (1000 = 1 secondo)

WIFIAP

Sintassi

WIFIAP <ssid_expr> <password_expr>

- <ssid_expr> e <password_expr> accettano:
  - testo **senza virgolette**: WIFIAP MyAP myap_pass
  - **variabili stringa**: WIFIAP APSSID$ APPASS$
  - **espressioni stringa**: WIFIAP LEFT$(APSSID$,5)+"_AP" APPASS$
  - (compatibile) **tra virgolette**: WIFIAP "My AP" "super pass"
- **Password AP**: minimo **8 caratteri** (requisito WPA2 dell'ESP).

Come sopra: senza virgolette, gli spazi separano i due argomenti. Per nomi/password con spazi, usa virgolette o variabili/espressioni.

Descrizione

Avvia un **Access Point** Wi-Fi sull'ESP32 (modalità **WIFI_AP**).
Utile per creare una rete locale autonoma per pagine di configurazione (es. ora/data) o per inviare comandi BASIC (GPIO, ecc.).

Output/effetti

- In caso di successo, l'AP è attivo.
- Puoi leggere l'IP dell'AP con:
  - PRINT IPAP

Esempi pratici

Esempio 1 – AP semplice

```
10 WIFIAP MyAP myap_pass
RUN
```

Output atteso:
Rete Wi-Fi locale creata.

Esempio 2 – AP con variabili

```
10 LET APSSID$="ConfigESP"
20 LET APPASS$="config123"
30 WIFIAP APSSID$ APPASS$
RUN
```

Esempio 3 – AP con espressione

```
10 LET BASE$="Demo"
20 WIFIAP BASE$+"_AP" "demopass1"
RUN
```

Esempio 4 – Pagina web sull’AP

```
10 WIFIAP MyAP myap_pass
20 HTML DEFAULT
30 HTMLOBJ "<h2>Esempio di pagina web</h2>"
60 HTMLSTART
RUN
```

Output atteso:
AP attivato e pagina web creata sull’IP locale (verifica con PRINT IPAP).

Esempio 5 – Pagina web che comanda GPIO su AP

```
10 PINMODE 2 OUTPUT NOPULL
20 WIFIAP MyAP myap_pass
30 HTML DEFAULT
40 HTMLOBJ "<h2>Controllo LED on board</h2>"
50 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 1\")'>Accendi
LED</button>"
60 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 0\")'>Spegni LED</button>"
70 HTMLSTART
RUN
```

Note e suggerimenti

- Password < 8 caratteri → errore.
- Se devi permettere **AP aperto** (senza password), si può prevedere una variante (WIFIAP SSID "") – dimmi se vuoi abilitarla.
- Per conoscere l’IP dell’AP: PRINT IPAP.

WIFI

Sintassi

WIFI <ssid_expr> <password_expr>
WIFI “ssid_expr” “password_expr”

- ssid_expr> e <password_expr> possono essere:
  - testo **senza virgolette**: WIFI MySSID MyPass
  - **variabili stringa**: WIFI SSID$ PASS$
  - **espressioni stringa** (concatenazioni/funzioni): WIFI LEFT$(SSID$,4)+"_GUEST" PASS$
  - (compatibile anche con la vecchia forma) **tra virgolette**: WIFI "My SSID" "My Pass"

Nota: se **non** usi virgolette o espressioni, lo **spazio** separa i due argomenti. Per SSID/password con spazi, usa virgolette o variabili/espressioni.

Descrizione

Connetta l’ESP32 a una rete Wi-Fi in modalità **station (WIFI_STA)**.
Utile per esporre pagine web sull’infrastruttura di rete esistente o sincronizzare impostazioni (es. data/ora) tramite rete.

Output/effetti

- In caso di successo, l’ESP si collega al Wi-Fi.
- Puoi leggere l’IP con:
  - PRINT IP (IP di stazione)
- Variabili utili esposte:
  - ESPMAC$ → MAC address della scheda

Esempi pratici

Esempio 1 – Connessione semplice

```
10 WIFI “MySSID” “MyPassword”
RUN
```

Output atteso:
Connessione a Wi-Fi attivata.

Esempio 2 – Connessione con variabili

```
10 LET SSID$="Casa Mia"
20 LET PASS$="qwerty123"
30 WIFI SSID$ PASS$
RUN
```

Output atteso:
Connessione a Wi-Fi attivata.

Esempio 3 – Connessione con espressione

```
10 LET BASE$="Ufficio"
20 WIFI BASE$+"_GUEST" "passguest"
RUN
```

Esempio 4 – Pagina web raggiungibile via IP

```
10 WIFI “MySSID” “MyPassword”
20 HTML DEFAULT
30 HTMLOBJ "<h2>Esempio di pagina web</h2>"
60 HTMLSTART
RUN
```

Output atteso:
Connessione a Wi-Fi attivata e pagina web creata sull’indirizzo IP (verifica con PRINT IP).

Esempio 5 – Pagina web che comanda GPIO

```
10 PINMODE 2 OUTPUT NOPULL
20 WIFI “MySSID” “MyPassword”
30 HTML DEFAULT
40 HTMLOBJ "<h2>Controllo LED on board</h2>"
50 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 1\")'>Accendi
LED</button>"
60 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 0\")'>Spegni LED</button>"
70 HTMLSTART
RUN
```

Note e suggerimenti

- Timeout di connessione ~10 s.
- Se PRINT IP restituisce 0.0.0.0, la connessione non è andata a buon fine.
- Per SSID/password con spazi, usa virgolette o variabili/espressioni (es. SSID$="Rete con spazi" → WIFI SSID$ PASS$).

WIFICONNECTED

Sintassi:
WIFICONNECTED

Descrizione:
Restituisce 1 se il dispositivo è connesso a una rete Wi-Fi tramite il comando WIFI, altrimenti 0.
Può essere utilizzato all'interno di PRINT, IF, o in assegnazioni di variabili.

Esempi pratici

Esempio 1 – Stampare lo stato della connessione

```
10 WIFI "ssid" "password"
20 WAIT 2000
30 PRINT WIFICONNECTED
RUN
```

Output atteso:

```
1
```

Esempio 2 – Condizione con IF

```
10 IF WIFICONNECTED = 0 THEN PRINT "NON CONNESSO"
RUN
```

Output atteso:

```
NON CONNESSO
```

Esempio 3 – LET

```
10 LET A = WIFICONNECTED
20 PRINT A
RUN
```

Output atteso:

```
SE CONNESSO 1 SE NON CONNESSO 0
```

Nota:
• Funziona dopo WIFI
• Restituisce 1 (connesso) o 0 (non connesso)
• Compatibile con PRINT, LET, IF

WIFIAPCONNECTED

Sintassi:
WIFIAPCONNECTED

Descrizione:
Restituisce 1 se il dispositivo ha crearo una rete Access Point tramite il comando WIFIAP, altrimenti 0.
Può essere utilizzato all'interno di PRINT, IF, o in assegnazioni di variabili.

Esempi pratici

Esempio 1 – Monitorare la connessione a un AP

```
10 WIFIAP "Basic32" "password"
20 IF WIFIAPCONNECTED = 1 THEN PRINT "RETE CREATA"
RUN
```

Output atteso:

```
RETE CREATA
```

Esempio 2 – Visualizzare stato AP

```
10 PRINT WIFIAPCONNECTED
RUN
```

Output atteso:

```
0 oppure 1
```

Nota:
• Funziona dopo WIFIAP
• Restituisce 1 se la rete Access Point è stata creata correttamante
• Utile per attivare azioni alla connessione

WIFIDISCONNECT

Sintassi:
WIFIDISCONNECT

Descrizione:
Disconnette il dispositivo dalla rete Wi-Fi precedentemente connessa con WIFI.
Restituisce sempre 0 e può essere usato anche in PRINT o IF O LET.

Esempi pratici

Esempio 1 – Disconnettere il Wi-Fi

```
10 WIFIDISCONNECT
20 PRINT "DISCONNESSO: "; WIFIDISCONNECT
RUN
```

Output atteso:

```
DISCONNESSO: 0
```

Esempio 2 – Uso con condizione

```
10 IF WIFIDISCONNECT = 0 THEN PRINT "OK"
RUN
```

Output atteso:

```
OK
```

Esempio 3 – Uso LET

```
10 LET A = WIFICONENCTED
RUN
```

Output atteso:

```
0
```

Nota:
• Disconnette solo la rete WIFI, non WIFIAP
• Restituisce 0 per compatibilità con IF e PRINT e LET
• Utile per passare tra reti o riavviare la connessione

WIFIAPDISCONNECT

Sintassi:
WIFIAPDISCONNECT

Descrizione:
Disattiva la modalità Access Point attiva, creata con WIFIAP.
Restituisce sempre 0, compatibile con IF e PRINT O LET.

Esempi pratici

Esempio 1 – Disattivare la rete Wi-Fi AP

```
10 WIFIAPDISCONNECT
20 PRINT "AP SPENTO: "; WIFIAPDISCONNECT
RUN
```

Output atteso:

```
AP SPENTO: 0
```

Esempio 2 – Condizione per verifica

```
10 IF WIFIAPDISCONNECT = 0 THEN PRINT "AP DISATTIVATO"
RUN
```

Output atteso:

```
AP DISATTIVATO
```

Esempio 3 – LET

```
10 LET A = WIFIAPCONNECTED
RUN
```

Output atteso:

```
0
```

Nota:
• Disattiva la rete AP
• Restituisce 0 come conferma
• Utile per risparmiare energia o sicurezza

WIRE

Sintassi

WIRE <sda> <scl>
WIRE AUTO <sda> <scl>
WIRE OFF

Descrizione

Il comando **WIRE** configura il bus I2C del sistema BASIC e gestisce sia l'inizializzazione immediata sia l'attivazione automatica all'avvio dell'ESP32.

1. WIRE <sda> <scl>

- Imposta i pin SDA e SCL.
- Inizializza immediatamente il bus I2C.
- La configurazione **non** viene salvata automaticamente per l'avvio.
- Tutti i comandi I2C del BASIC richiedono che WIRE sia stato eseguito almeno una volta.

2. WIRE AUTO <sda> <scl>

- Imposta i pin SDA e SCL.
- Inizializza immediatamente il bus I2C.
- **Salva** la configurazione, che verrà automaticamente riattivata ad ogni avvio dell'ESP32.
- Rimane valida anche dopo un reset o power cycle.

3. WIRE OFF

- Disattiva l'inizializzazione automatica del bus I2C all'avvio.
- Se il bus è attivo in quel momento, rimane attivo fino al riavvio.
- Non modifica i pin attualmente configurati, elimina solo l'autostart.

Esempi pratici

Esempio 1 – Configurazione minima (runtime)

```
10 WIRE 21 22
20 SCANI2C
RUN
```

Output possibile:

```
Scanning I2C bus...
```

Found device at 0x3C (60)
Total I2C devices: 1

Esempio 2 – Configurare e inizializzare l’RTC

```
10 WIRE 21 22
20 INITRTC 0x68
30 PRINT "RTC ok!"
RUN
```

Esempio 3 – Auto-configurazione I2C al boot

```
10 WIRE AUTO 21 22
20 PRINT "I2C pronto!"
RUN
```

Dopo un riavvio dell’ESP32, il bus verrà inizializzato automaticamente con gli stessi pin.

Esempio 4 – Disattivare l’I2C automatico

```
10 WIRE OFF
20 PRINT "Auto-I2C disattivato."
RUN
```

Alla prossima accensione WIRE NON verrà eseguito automaticamente.

Esempio 5 – Errore se WIRE manca

```
10 INITRTC 0x68
RUN
```

Output atteso:

RUNTIME ERROR: INITRTC: WIRE must be called first

Note

- Il comando **non usa virgole** → solo spazi tra argomenti.
- SDA e SCL devono essere numeri validi dei pin della board ESP32.
- WIRE deve essere eseguito **prima** di SCANI2C, INITRTC o qualsiasi sensore/attuatore I2C.
- WIRE AUTO salva la configurazione in /prefs.cfg e la riapplica automaticamente al boot.

- WIRE OFF non rimuove i pin configurati, disattiva solo la modalità automatica.
- È consigliato chiamare WIRE (o WIRE AUTO) all’inizio del programma per evitare comportamenti inattesi.

WS BRIGHTNESS

Sintassi
WS BRIGHTNESS value

Descrizione
Imposta la luminosità globale (0–255) della **striscia selezionata**.
Si compone con WS LBRIGHT (per-LED): il valore finale di ogni canale è:

output = raw * (LBRIGHT / 255) * (BRIGHTNESS / 255)

Esempio – Dimmer su rosso

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 0 0
40 WS BRIGHTNESS 40
50 WS SHOW
```

Esempio – Effetto “respiro”

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 0 120 255
40 B = 0
50 WS BRIGHTNESS B
60 WS SHOW
70 B = B + 5
80 IF B > 255 THEN GOTO 120
90 WAIT 20
100 GOTO 50
120 WS BRIGHTNESS B
130 WS SHOW
140 B = B - 5
150 IF B < 0 THEN GOTO 40
160 WAIT 20
170 GOTO 120
```

Note

- 0 = spento, 255 = massima luminosità globale.
- La luminosità per-LED (WS LBRIGHT) e quella globale (WS BRIGHTNESS) si moltiplicano fra loro:
  output = raw * (LBRIGHT/255) * (BRIGHTNESS/255)
- WS BRIGHTNESS non invia subito i dati: serve un WS SHOW (o WS SHOW ALL).

WS BUS BRIGHTNESS

Sintassi
WS BUS BRIGHTNESS value

Descrizione
Imposta la luminosità globale (0–255) **uguale per tutte le strisce** nel BUS.
Ogni strip del BUS riceve lo stesso BRIGHTNESS, mantenendo i propri LBRIGHT per-LED.

Esempio – Dimmer globale bus

```
10 WS NEW 0 13 5
20 WS NEW 1 27 5
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 255 0
50 WS BUS BRIGHTNESS 50
60 WS BUS SHOW
```

Esempio – Fade su tutto il bus

```
10 WS NEW 0 13 5
20 WS NEW 1 27 5
30 WS BUS DEFINE 0 1
40 B = 0
50 WS BUS BRIGHTNESS B
60 WS BUS SHOW
70 B = B + 10
80 IF B > 180 THEN B = 180
90 WAIT 30
100 IF B < 180 THEN GOTO 50
```

Note

- Richiede che il BUS sia stato definito con WS BUS DEFINE.
- Compatibile con WS BUS LBRIGHT: per ogni LED virtuale valgono sia LBRIGHT che la BRIGHTNESS di strip.

WS BUS COUNT

Sintassi
WS BUS COUNT var

Descrizione
Scrive in var il numero totale di LED nel BUS, cioè la somma dei LED di tutte le strip incluse in WS BUS DEFINE.

Esempio – Stampa count bus

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 PRINT "BUS LED=";N
```

Esempio – Runner su tutto il bus

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 POS = 0
60 WS BUS FILL 0 0 0
70 WS BUS SET POS 255 255 255
80 WS BUS SHOW
90 POS = POS + 1
100 IF POS >= N THEN POS = 0
110 WAIT 40
120 GOTO 60
```

Note

- Se ricrei/elimini strip (WS NEW, WS DELETE, WS DELETE ALL), esegui nuovamente WS BUS DEFINE prima di riusare il BUS.

WS BUS DEFINE

Sintassi
WS BUS DEFINE id1 id2 [id3 ...]

Descrizione
Definisce l'ordine delle strisce concatenate nel BUS.
Gli indici "virtuali" del BUS vanno da 0 a N-1, dove N è il totale dei LED (WS BUS COUNT).
L'ordine è dato dalla sequenza degli id passati (id1, id2, ...).

Esempio – Due strisce in serie

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 0 255
50 WS BUS SHOW
```

Esempio – Tre blocchi colorati

```
10 WS NEW 0 13 4
20 WS NEW 1 27 4
30 WS NEW 2 26 4
40 WS BUS DEFINE 2 0 1
50 WS BUS COUNT N
60 I = 0
70 IF I >= N THEN GOTO 120
80 IF I < N/3 THEN WS BUS SET I 0 255 0
90 IF I >= N/3 THEN IF I < (2*N)/3 THEN WS BUS SET I 0 0 255
100 IF I >= (2*N)/3 THEN WS BUS SET I 255 0 0
110 I = I + 1
115 GOTO 70
120 WS BUS SHOW
```

Note

- Tutti gli id indicati devono esistere (WS NEW) e la strip deve essere ancora valida.
- Puoi ridefinire il BUS quando vuoi: sovrascrive la definizione precedente.

WS BUS FILL

Sintassi
WS BUS FILL R G B

Descrizione
Imposta **tutti i LED virtuali** del BUS al colore (R,G,B) sui buffer interni (senza inviare).
L'aggiornamento reale avviene con WS BUS SHOW.

Esempio – On/Off su tutto il bus

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 255 255 255
50 WS BUS SHOW
60 WAIT 300
70 WS BUS FILL 0 0 0
80 WS BUS SHOW
```

Esempio – Alterna due palette

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 255 120 0
50 WS BUS SHOW
60 WAIT 300
70 WS BUS FILL 0 80 255
80 WS BUS SHOW
90 WAIT 300
100 GOTO 40
```

Note

- Utile per inizializzare il BUS a un colore base prima di applicare effetti con WS BUS SET e WS BUS LBRIGHT.

WS BUS LBRIGHT

Sintassi
WS BUS LBRIGHT idx value

Descrizione
Imposta la luminosità per-LED (0–255) del **LED virtuale** idx nel BUS.
Il LED è scelto sull'indice concatenato, in base alla definizione di WS BUS DEFINE.

Esempio – Puntatore con coda su BUS

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 HEAD = 0
60 WS BUS FILL 255 0 0
70 I = 0
80 IF I >= N THEN GOTO 110
90 WS BUS LBRIGHT I 10
100 I = I + 1
105 GOTO 80
110 WS BUS LBRIGHT HEAD 255
120 T = 1
130 IF T > 5 THEN GOTO 170
140 K = HEAD - T
150 IF K < 0 THEN K = K + N
160 WS BUS LBRIGHT K 255 - T * 40
165 T = T + 1
167 GOTO 130
170 WS BUS SHOW
180 HEAD = HEAD + 1
190 IF HEAD >= N THEN HEAD = 0
200 WAIT 50
210 GOTO 60
```

Esempio – Bargraph su BUS

```
10 WS NEW 0 13 6
20 WS NEW 1 27 6
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 LEVEL = 0
60 WS BUS FILL 0 60 0
70 I = 0
80 IF I >= N THEN GOTO 120
90 IF I < LEVEL THEN WS BUS LBRIGHT I 255
```

```
100 IF I >= LEVEL THEN WS BUS LBRIGHT I 20
110 I = I + 1
115 GOTO 80
120 WS BUS SHOW
130 LEVEL = LEVEL + 1
140 IF LEVEL > N THEN LEVEL = 0
150 WAIT 100
160 GOTO 60
```

Note

- Si compone con WS BUS BRIGHTNESS:
  output = raw * (LBRIGHT/255) * (BRIGHTNESS/255) sul LED virtuale corrispondente.

WS BUS SET

Sintassi
WS BUS SET idx R G B

Descrizione
Imposta il colore del LED virtuale idx nel BUS (indice concatenato), sui buffer interni. L'aggiornamento reale avviene con WS BUS SHOW.

Esempio – Runner sul BUS

```
10 WS NEW 0 13 5
20 WS NEW 1 27 7
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 POS = 0
60 WS BUS FILL 0 0 0
70 WS BUS SET POS 255 255 255
80 WS BUS SHOW
90 POS = POS + 1
100 IF POS >= N THEN POS = 0
110 WAIT 40
120 GOTO 60
```

Esempio – Segmenti alternati

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS COUNT N
50 LEN = 4
60 POS = 0
70 WS BUS FILL 0 0 0
80 I = 0
90 IF I >= LEN THEN GOTO 130
100 IDX1 = POS + I
110 IF IDX1 >= N THEN IDX1 = IDX1 - N
120 WS BUS SET IDX1 255 0 0
125 I = I + 1
127 GOTO 90
130 I = 0
140 IF I >= LEN THEN GOTO 190
150 IDX2 = POS + 8 + I
160 IF IDX2 >= N THEN IDX2 = IDX2 - N
170 WS BUS SET IDX2 0 0 255
180 I = I + 1
185 GOTO 140
```

```
190 WS BUS SHOW
200 POS = POS + 1
210 IF POS >= N THEN POS = 0
220 WAIT 60
230 GOTO 70
```

Note

- È l’equivalente di WS SET, ma lavora sull’indice **virtuale** del BUS, non sull’indice locale della singola strip.

WS BUS SHOW

Sintassi
WS BUS SHOW

Descrizione
Invia i buffer di **tutte le strisce presenti nel BUS**, applicando per ogni LED:

- colore raw (da WS BUS FILL / WS BUS SET)
- luminosità per-LED (WS BUS LBRIGHT)
- luminosità globale (WS BUS BRIGHTNESS)

Esempio – Invio dopo FILL

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS BUS DEFINE 0 1
40 WS BUS FILL 0 255 0
50 WS BUS SHOW
```

Esempio – Invio dopo molte SET

```
10 WS NEW 0 13 4
20 WS NEW 1 27 4
30 WS BUS DEFINE 0 1
40 WS BUS SET 0 255 0 0
50 WS BUS SET 7 0 0 255
60 WS BUS SHOW
```

Note

- In modalità BUS è preferibile usare WS BUS SHOW invece di WS SHOW ALL.

WS CLEAR

Sintassi
WS CLEAR

Descrizione
Spegne subito tutti i LED della **striscia selezionata**:

- azzera i colori nei buffer (raw R,G,B = 0)
- invia immediatamente l’aggiornamento alla strip

Esempio – Reset strip

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS CLEAR
```

Esempio – Flash

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 255 255
40 WS SHOW
50 WAIT 80
60 WS CLEAR
```

Note

- Dopo WS CLEAR la strip resta spenta finché non imposti di nuovo i colori e non esegui WS SHOW.
- WS CLEAR non modifica WS BRIGHTNESS né i valori WS LBRIGHT.

WS COUNT

Sintassi
WS COUNT var

Descrizione
Scrive in var il numero di LED della **striscia selezionata**.

Esempio – Stampa count strip

```
10 WS NEW 0 13 10
20 WS SELECT 0
30 WS COUNT N
40 PRINT "LED strip 0 = ";N
```

Esempio – Runner sicuro

```
10 WS NEW 0 13 10
20 WS SELECT 0
30 WS COUNT N
40 I = 0
50 WS FILL 0 0 0
60 WS SET I 255 255 255
70 WS SHOW
80 I = I + 1
90 IF I >= N THEN I = 0
100 WAIT 50
110 GOTO 50
```

Note

- Se nessuna strip è selezionata (WS SELECT), il comando genera errore a runtime.

WS DELETE

Sintassi
WS DELETE id

Descrizione
Rimuove la striscia id e libera la memoria associata:

- se era nel BUS, viene rimossa dal BUS (che viene ricalcolato)
- se era la strip selezionata, la selezione viene azzerata

Esempio – Cancella una strip

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS DELETE 1
```

Esempio – Ricrea con altro count

```
10 WS NEW 0 13 8
20 WS DELETE 0
30 WS NEW 0 13 16
40 WS SELECT 0
50 WS FILL 0 0 255
60 WS SHOW
```

Note

- Se vuoi essere sicuro di spegnere prima la strip reale, usa:
  WS SELECT id + WS CLEAR, poi WS DELETE id.

WS DELETE ALL

Sintassi
WS DELETE ALL

Descrizione
Rimuove tutte le strip create, svuota il BUS e azzera la strip selezionata.

Esempio – Reset totale

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS DELETE ALL
```

Esempio – Riparti da zero

```
10 WS DELETE ALL
20 WS NEW 0 13 12
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SHOW
```

Note

- Dopo WS DELETE ALL devi ricreare le strip con WS NEW e, se ti serve il bus, ridefinirlo con WS BUS DEFINE.

WS FILL

Sintassi
WS FILL R G B

Descrizione
Imposta tutti i LED della **striscia selezionata** al colore (R,G,B) nei buffer interni.
Per aggiornare fisicamente la strip serve un WS SHOW (o WS SHOW ALL).

Esempio – Rosso tenue

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 0 0
40 WS BRIGHTNESS 30
50 WS SHOW
```

Esempio – Blink

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 255 255 255
40 WS SHOW
50 WAIT 150
60 WS FILL 0 0 0
70 WS SHOW
80 WAIT 150
90 GOTO 30
```

Note

- Per lavorare su singoli LED usa WS SET.

WS LBRIGHT

Sintassi
WS LBRIGHT idx value

Descrizione
Imposta la luminosità per-LED (0–255) del LED idx sulla striscia selezionata (solo buffer). È un fattore di scala che non cambia il colore, solo l'intensità relativa del LED.

Esempio – Scala di intensità

```
10 WS NEW 0 13 16
20 WS SELECT 0
30 WS FILL 0 120 255
40 I = 0
50 IF I >= 16 THEN GOTO 80
60 WS LBRIGHT I I*16
70 I = I + 1
75 GOTO 50
80 WS SHOW
```

Esempio – Puntatore con coda

```
10 WS NEW 0 13 12
20 WS SELECT 0
30 COUNT = 12
40 POS = 0
50 WS FILL 255 0 0
60 I = 0
70 IF I >= COUNT THEN GOTO 100
80 WS LBRIGHT I 10
90 I = I + 1
95 GOTO 70
100 WS LBRIGHT POS 255
110 T = 1
120 IF T > 5 THEN GOTO 160
130 K = POS - T
140 IF K < 0 THEN K = K + COUNT
150 WS LBRIGHT K 255 - T * 40
155 T = T + 1
157 GOTO 120
160 WS SHOW
170 POS = POS + 1
180 IF POS >= COUNT THEN POS = 0
190 WAIT 50
200 GOTO 50
```

Note

- Non modifica il colore “raw”, solo la sua intensità.
- Si compone con WS BRIGHTNESS secondo:
  output = raw * (LBRIGHT/255) * (BRIGHTNESS/255)

WS NEW (NeoPixel WS2812B)

Sintassi
WS NEW id pin count

Descrizione
Crea/inizializza una striscia NeoPixel con identificatore id, collegata a pin e con count LED.
Se id esiste già, la striscia viene ricreata con i nuovi parametri.

Esempio – Due strip

10 WS NEW 0 13 8
20 WS NEW 1 27 12

Esempio – Ricrea con più LED

10 WS NEW 0 13 8
20 WS NEW 0 13 16

Note (HW)

- Alimentazione 5 V **stabile**, GND in comune con l'ESP32.
- Consigliata resistenza ~330 Ω in serie sulla linea DATA.
- Consigliato condensatore 470–1000 µF vicino all'ingresso 5 V della strip.
- Per strip lunghe, inietta 5 V in più punti lungo la striscia.

WS SELECT

Sintassi
WS SELECT id

Descrizione
Seleziona la striscia su cui operano tutti i comandi WS ... **non BUS** (FILL, SET, SHOW, BRIGHTNESS, LBRIGHT, COUNT, CLEAR, ecc.).

Esempio – Lavorare su id=1

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 1
40 WS FILL 0 0 255
50 WS SHOW
```

Esempio – Alternare due strip

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SHOW
60 WS SELECT 1
70 WS FILL 0 0 255
80 WS SHOW
```

Note

- Se nessuna strip è selezionata (WS SELECT non chiamato o strip rimossa), i comandi WS ... che richiedono una strip falliscono con errore.

WS SET

Sintassi
WS SET idx R G B

Descrizione
Imposta il colore “raw” del LED idx sulla striscia selezionata (solo buffer).
Per aggiornare la strip serve un WS SHOW (o WS SHOW ALL).

Esempio – Arcobaleno manuale 7 LED

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS SET 0 255 0 0
40 WS SET 1 255 127 0
50 WS SET 2 255 255 0
60 WS SET 3 0 255 0
70 WS SET 4 0 180 255
80 WS SET 5 0 0 255
90 WS SET 6 148 0 211
100 WS SHOW
```

Esempio – Theater chase (periodo 3)

```
10 WS NEW 0 13 12
20 WS SELECT 0
30 PH = 0
40 WS FILL 0 0 0
50 I = 0
60 IF I >= 12 THEN GOTO 110
70 T = I + PH
80 IF T >= 3 THEN T = T - 3
90 IF T = 0 THEN WS SET I 255 100 0
100 I = I + 1
105 GOTO 60
110 WS SHOW
120 PH = PH + 1
130 IF PH >= 3 THEN PH = 0
140 WAIT 80
150 GOTO 40
```

Note

- L’indice idx va da 0 a count-1, dove count è il numero di LED della strip (WS COUNT).

WS SHOW

Sintassi

WS SHOW
WS SHOW ALL

Descrizione

- WS SHOW → invia i buffer della **striscia selezionata**.
- WS SHOW ALL → invia i buffer di **tutte le strisce create** (non BUS).

In entrambi i casi vengono applicati WS LBRIGHT per-LED e WS BRIGHTNESS globale.

Esempio – Strip corrente

```
10 WS NEW 0 13 8
20 WS SELECT 0
30 WS FILL 0 255 0
40 WS SHOW
```

Esempio – Tutte le strip

```
10 WS NEW 0 13 8
20 WS NEW 1 27 8
30 WS SELECT 0
40 WS FILL 255 0 0
50 WS SELECT 1
60 WS FILL 0 0 255
70 WS SHOW ALL
```

Note

- In modalità BUS, preferisci WS BUS SHOW invece di WS SHOW ALL.

INDICE