

BASIC32

BASIC IS BACK ON BOARD



Versione 081025

<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. 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** (qualsiasi modello 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 un **firmware già compilato**. Non è necessario usare l'Arduino IDE: basta scaricare il file .bin e flasharlo direttamente nella memoria del dispositivo.

1. Requisiti

- Scheda **ESP32** (qualsiasi modello con supporto SPIFFS e SD opzionale)
 - **Cavo USB** per collegare l'ESP32 al PC
 - **Tool per flash firmware:**
 - **Basic32 Terminal** (Windows)
 - [esptool.py](#) (Linux/macOS/Windows via Python)
 - Terminale seriale (es. Basic32 Terminal, PuTTY, TeraTerm, Arduino Serial Monitor)
-

2. File da scaricare

- Basic32.bin → firmware precompilato (fornito su github del progetto)
 - Eventuali file .bas di esempio (opzionali)
-

3. Flash del Firmware su ESP32

Metodo 1: con Basic32 Terminal (windows)

1. Installa Basic32 Terminal (se non l'hai già fatto):

<https://github.com/Ferrazzi/Basic32/tree/main/Basic32Terminal>

2. Collega l'ESP32 e dalle icone seleziona quella per flashare il firmware
3. Seleziona se flasharlo da file o tramite internet, seleziona il tuo modello di ESP32 e clicca su Flash

Metodo 2: con esptool.py (multiplatforma)

1. Installa esptool.py (se non l'hai già fatto):

```
pip install esptool
```

2. Collega l'ESP32 e identifica la porta seriale (es: COM3 su Windows o /dev/ttyUSB0 su Linux)
3. Flasha il firmware con questo comando (modifica la porta e il percorso se necessario):

```
esptool.py --chip esp32 --port COM3 --baud 460800 write_flash -z 0x10000 Basic32.bin
```

4. 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
```

5. Utilizzo SPIFFS

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

```
ESAVE "programma.bas"  
ELOAD "programma.bas"
```

6. 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

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

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

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 $\sim V_{cc}/2$). 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 $\sqrt{\text{media}(x^2)}$.

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 \geq limit).
- Con **BELOW**: salta se corrente \leq 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

Sintassi:

```
ADC CAL REF <volt>
ADC CAL GAIN <fattore>
ADC CAL OFFSET <volt>
ADC CAL MEASURE <pin> <volt_noto> [campioni]
ADC CAL STATUS
ADC CAL RESET
```

Descrizione:

ADC CAL gestisce la **calibrazione dell'ADC** usata da VREAD e RREAD.

- REF imposta il riferimento (es. 3.30 V).
 - GAIN aggiusta il guadagno moltiplicativo.
 - OFFSET aggiunge un offset in volt.
 - MEASURE calcola automaticamente il GAIN misurando un **volt noto** applicato al pin (opzionale media su N campioni).
 - STATUS mostra i parametri correnti.
 - RESET ripristina i default (ESP32: REF=3.30, GAIN=1.0, OFFSET=0.0).
-

Esempi pratici

Esempio 1 – Impostare solo il riferimento

```
ADC CAL REF 3.30
```

→ Usa 3.30 V come riferimento ADC.

Esempio 2 – Calibrazione automatica con tensione nota su pin 34

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

→ Misura la tensione media su GPIO34 (32 campioni) e calcola il GAIN per farla combaciare con 3.300 V.

Esempio 3 – Impostare manualmente gain e offset

```
ADC CAL GAIN 1.015
ADC CAL OFFSET -0.010
```

→ Aumenta la lettura del 1.5% e sottrae 10 mV come offset.

Esempio 4 – Verificare i parametri

```
ADC CAL STATUS
```

→ Stampa i valori correnti su Serial.

Esempio 5 – Ripristinare i default

ADC CAL RESET

→ Torna ai valori di fabbrica.

Output atteso:

- Per STATUS (su Serial, es.):

ADC REF=3.300000 GAIN=1.000000 OFFSET=0.000000

- In caso di errore:

?ADC CAL ERROR

known_volts must be > 0

Nota:

- MEASURE richiede che sul pin sia presente **una tensione reale e stabile** uguale al valore passato (es. 3.300 V).
- REF, GAIN e OFFSET si applicano **sempre** a VREAD e RREAD.
- Per ESP32 si consiglia l'uso di pin **ADC1** (es. 34, 35, 32, 33).
- Non applicare mai >3.3 V ai pin ADC dell'ESP32.

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:

Esempio 4 – Analisi input da tastiera (GET + ASC)

```
10 PRINT "PREMI UN TASTO:"  
20 GET K$  
30 PRINT "CODICE ASCII: "; ASC(K$)  
RUN
```

Output atteso:

Mostra il codice del tasto premuto.

Esempio 5 – 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 il comando **AT** (aggiunge CRLF), poi legge **una riga** di risposta entro timeout ms (se omissso usa il timeout di BT TIMEOUT). La risposta va in var\$.

Esempi pratici

Esempio 1 – Test AT

```
10 BT AT "AT" RESP R$ 1000
20 PRINT R$
RUN
```

Output atteso:

OK

Esempio 2 – Leggere il nome

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

Output atteso:

+NAME: BASIC32

Nota:

- Richiede AT mode attivo (usa BT ATMODE ON o KEY alto all'avvio)
- Timeout in ms; se mancante usa BT TIMEOUT

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 16 17 9600 4
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 AVAILABLE N
20 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 BT.

Esempi pratici

```
10 BT FLUSH
20 BT READLN S$
30 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 l'HC-05/HC-06 in **modo dati** alla velocità baud. Opzionalmente imposta keyPin per AT mode.

Esempi pratici

Esempio 1 – Base

```
10 BT INIT 16 17 9600
20 PRINT "OK"
RUN
```

Output atteso:

OK

Esempio 2 – Con KEY su GPIO4

```
10 BT INIT 16 17 9600 4
20 PRINT "READY"
RUN
```

Output atteso:

READY

Nota:

- Scegli pin compatibili con la tua board ESP32

BT PRINT

Sintassi:

BT PRINT valore

Descrizione:

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

Esempi pratici

```
10 LET A=42
20 BT PRINT A
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 omissso, legge tutto quello che c'è) e li mette in var\$.

Esempi pratici

Esempio 1 – Leggere tutto

```
10 BT READ S$
20 PRINT S$
RUN
```

Output atteso:

HELLO WORLD

Esempio 2 – 5 byte

```
10 BT READ P$ 5
20 PRINT P$
RUN
```

Output atteso:

HELLO

Nota:

- Non attende newline; se non ci sono dati, var\$ è vuota

BT READLN

Sintassi:

BT READLN var\$

Descrizione:

Legge fino a **LF** (\n), ignorando \r. Se non trova LF, restituisce quanto presente (anche stringa vuota).

Esempi pratici

```
10 BT READLN CMD$
20 IF LEN(CMD$)=0 THEN PRINT "NESSUNA RIGA"
RUN
```

Output atteso:

NESSUNA RIGA

Nota:

- Ideale per protocolli a riga (\n come terminatore)

BT SEND

Sintassi:

BT SEND "testo"

BT SEND var\$

Descrizione:

Invia la stringa (letterale o var\$) **così com'è** sulla UART BT.

Esempi pratici

```
10 BT SEND "READY\r\n"
```

```
RUN
```

Output atteso:

(nessun output console; la stringa va su BT)

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 ATMODE ON
20 BT SETBAUD 9600
30 BT ATMODE OFF
40 BT END
50 BT INIT 16 17 9600
60 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)

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 "file"  
COPY FILE$  
COPY "**"
```

Descrizione

Il comando **COPY** copia i file **dalla SPIFFS alla scheda SD**.
Accetta:

- Nomi di file in virgolette
- Variabili stringa (FILE\$) o espressioni
- "*" per copiare **tutti i file** dalla SPIFFS alla SD

Se il file esiste già in SD, viene sovrascritto.

Esempi

Esempio 1 – Copia un file specificato da variabile

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

Esempio 2 – Copia tutti i file presenti in SPIFFS

```
10 COPY "**"  
20 PRINT "Tutti i file copiati dalla SPIFFS alla SD"
```

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 "file"

(o DEL F\$)

Sintassi:

DEL "nomefile"
DEL variabile\$

Descrizione:

Il comando DEL cancella un file dalla **memoria SD**.

Può accettare sia un **nome file scritto direttamente** tra virgolette, sia una **variabile stringa** che contiene il nome del file.

Non produce errori se il file non esiste.

Esempi pratici

Esempio 1 – Eliminare file con nome diretto

```
DEL "prog1.bas"
```

Output atteso:

(Il file viene eliminato senza messaggi)

Esempio 2 – Usare una variabile per specificare il file

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

Output atteso:

(Il file viene eliminato senza messaggi)

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

Sintassi:

DELVAR "file"

DELVAR "file" "chiave"

Descrizione:

Il comando DELVAR elimina un file JSON intero o una **singola chiave** all'interno del file.

- Senza secondo parametro: elimina l'intero file
- Con la chiave: elimina solo quella voce

☐ Agisce solo sulla **SD**
Per SPIFFS usa EDELVAR

Esempio 1 – Eliminare una chiave dal file:

```
10 DELVAR "config.json" "NOME"
```

Esempio 2 – Eliminare completamente il file:

```
10 DELVAR "config.json"
```

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 OLED SET 21 22 0x3C 128 64 0  
30 DEV DISPLAY OLED  
40 DEV ON  
RUN
```

→ Alla successiva accensione l'ESP32 partirà direttamente in modalità developer con OLED 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 CURSOR

Sintassi:

DEV CURSOR ON
DEV CURSOR OFF

Descrizione:

Abilita o disabilita il cursore lampeggiante sul display developer.
Può essere utile per distinguere se si è in modalità input.

Esempi pratici:

DEV CURSOR ON

→ Mostra il cursore lampeggiante.

DEV CURSOR OFF

→ Nasconde il cursore.

DEV DISPLAY

Sintassi

DEV DISPLAY OLED
DEV DISPLAY ILI
DEV DISPLAY LCD

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.

Esempi pratici

Esempio 1 – Usa l'OLED

```
10 DEV DISPLAY OLED
20 DEV CLEAR
30 PRINT "Benvenuto su OLED"
RUN
```

Esempio 2 – Seleziona ILI9341

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

Esempio 3 – Seleziona LCD (caratteri)

```
10 DEV DISPLAY LCD
20 DEV CLEAR
30 PRINT "LCD operativo"
RUN
```

Note

- Puoi passare da un display all'altro con lo stesso comando: DEV DISPLAY OLED/ILI/LCD.
- Se il dispositivo non è configurato o non inizializza, il comando segnala errore e resta il display precedente.
- **Prestazioni:** su ILI/OLED, 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 LCD I2C SET

Sintassi:

DEV LCD I2C SET <addr> <cols> <rows> <sda> <scl>

Descrizione:

Configura un display LCD basato su adattatore I²C.

Richiede indirizzo I²C (addr), dimensioni (cols, rows) e i pin SDA/SCL.

Esempio pratico:

```
DEV LCD I2C SET 3C 16 2 21 22
DEV DISPLAY LCD
DEV ON
```

→ Configura un LCD I²C 16x2 con indirizzo 0x3C su SDA=21, SCL=22.

DEV LCD PAR SET

Sintassi:

DEV LCD PAR SET <rs> <en> <d4> <d5> <d6> <d7> <cols> <rows>

Descrizione:

Configura un display LCD in modalità parallela 4 bit.
Richiede i pin RS, EN, D4..D7 e le dimensioni.

Esempio pratico:

```
DEV LCD PAR SET 12 11 5 4 3 2 20 4
DEV DISPLAY LCD
DEV ON
```

→ Configura un LCD parallelo 20x4.

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 OLED SET

Sintassi:

DEV OLED SET <width> <height> <addr> <sda> <scl> <rst>

Descrizione:

Configura un display OLED SSD1306 su bus I²C.

Richiede pin SDA/SCL, indirizzo I²C, dimensioni e pin RST.

Esempio pratico:

```
DEV OLED SET 128 64 3C 21 22 0
DEV DISPLAY OLED
DEV ON
```

→ Configura un OLED 128x64 con indirizzo 0x3C.

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 OLED
DEV ON

→ Attiva la console su OLED 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 OFF

Sintassi:

DEV PS2 OFF

Descrizione:

Disattiva la tastiera PS/2 in modalità developer.

Esempi pratici

DEV PS2 OFF

Nota:

- Interrompe la lettura della tastiera PS/2.

DEV PS2 ON

Sintassi:

DEV PS2 ON

Descrizione:

Attiva la tastiera PS/2 per la console DEV.

Esempi pratici

DEV PS2 ON

Nota:

- Richiede configurazione dei pin con DEV PS2 SET.

DEV PS2 SET

Sintassi:

DEV PS2 SET <clkPin> <dataPin>

Descrizione:

Configura i pin per collegare una tastiera PS/2 al sistema.

La tastiera può essere usata per digitare direttamente i comandi BASIC sul display in dev mode.

Esempio pratico:

DEV PS2 SET 33 32

→ Abilita una tastiera PS/2 su CLK=33, DATA=32.

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 (OLED/ILI/LCD) 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/OLED 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 (OLED/ILI/LCD); non influisce sull'input da tastiera/PS2 o su altre periferiche.
- **Persistenza:** l'impostazione non viene salvata in config; metti DEV SERIAL ON/OFF nel tuo **AUTOEXEC**/script di avvio se vuoi applicarla automaticamente.
- **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 OLED/ILI/LCD.
- Gli indirizzi I²C sono in hex senza 0x.

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 SDHCLIBSHOW
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 SDHCLIBSHOW
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:

$\text{valore_calibrato} = \text{valore_grezzo} * \text{slope} + \text{offset}$

- 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:

DIM nome_array(n)

Descrizione:

Il comando DIM viene utilizzato per **dichiarare un array** (vettore) numerico.

L'array sarà indicizzato da 0 a n (incluso), quindi se fai DIM A(5), l'array avrà **6 elementi**: A(0) fino a A(5).

Gli array vengono inizializzati a 0 e possono contenere solo **valori numerici**.

Puoi avere più array nel programma, ciascuno con un nome diverso.

Esempi pratici

Esempio 1 – Dichiarare e usare un array

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

Output atteso:

```
A(0) = 5
A(1) = 10
A(2) = 15
```

Esempio 2 – Accesso diretto a un indice

```
10 DIM X(2)
20 X(1) = 99
30 PRINT X(1)
RUN
```

Output atteso:

```
99
```

Esempio 3 – Uso con variabili

→ Puoi accedere a un array usando una variabile come indice:

```
10 DIM N(5)
20 FOR I = 0 TO 5
```

```
30 N(I) = I * I
40 NEXT I
50 PRINT "N(3) = "; N(3)
RUN
```

Output atteso:

N(3) = 9

Esempio 4 – Errore comune evitabile

→ Se accedi a un indice **non dichiarato**, il programma può restituire errore o valore non valido:

```
10 DIM A(2)
20 A(5) = 10 ' ERRORE: 5 è fuori dai limiti
```

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/PULLEDOWN/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

Descrizione:

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

Se una **scheda SD è collegata e rilevata**, DIR mostrerà i file sulla SD.

Utile per visualizzare rapidamente i file disponibili da caricare, cancellare o rinominare.

Esempi pratici

Esempio – Elencare file in memoria

DIR

Output atteso (esempio):

```
program1.bas  
demo.bas
```

ECOPY

Sintassi

```
ECOPY "file"  
ECOPY FILE$  
ECOPY "**"
```

Descrizione

Il comando **ECOPY** copia i file **dalla scheda SD alla memoria SPIFFS**.

Accetta:

- Nomi di file in virgolette ("config.json")
- Variabili stringa (FILE\$) o espressioni stringa
- Il carattere "*" per copiare **tutti i file** dalla SD alla SPIFFS

Se il file esiste già in SPIFFS, viene sovrascritto.

Esempi

Esempio 1 – Copia un singolo file usando variabile

```
10 FILE$ = "config.json"  
20 ECOPY FILE$  
30 PRINT "File copiato in SPIFFS"
```

Esempio 2 – Copia tutti i file

```
10 ECOPY "**"  
20 PRINT "Tutti i file copiati dalla SD alla SPIFFS"
```

Esempio 3 – Usa un'espressione stringa

```
10 NAME$ = "init"  
20 ECOPY NAME$ + ".bas"  
30 PRINT "Copiato: "; NAME$; ".bas"
```

EDEL

Sintassi:

EDEL "nomefile"
EDEL variabile\$

Descrizione:

Il comando EDEL funziona come DEL, ma agisce **esclusivamente sulla memoria interna (SPIFFS), anche se è presente una scheda SD.**

È utile per assicurarsi di cancellare file **solo nella memoria interna**, senza ambiguità.

Esempi pratici

Esempio 1 – Eliminazione forzata da SPIFFS

```
EDEL "prog1.bas"
```

Esempio 2 – Usare variabile per specificare il file

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

Output atteso:

(Il file viene eliminato senza messaggi)

EDELVAR

Sintassi:

EDELVAR "file"
EDELVAR "file" "chiave"

Descrizione:

EDELVAR funziona come DELVAR, ma opera sulla **SPIFFS**.

Esempio 1 – Eliminare una chiave da prefs.json su SPIFFS:

```
10 EDELVAR "prefs.json" "USER"
```

Esempio 2 – Cancellare tutto il file:

```
10 EDELVAR "prefs.json"
```

EDIR

Sintassi:

EDIR

Descrizione:

Il comando EDIR forza la visualizzazione dei file contenuti **solo nella memoria interna SPIFFS**, anche se è presente una scheda SD.

Questo comando è utile quando si desidera **gestire separatamente** i file tra SPIFFS e SD.

Esempi pratici

Esempio – Mostrare solo i file su SPIFFS

EDIR

Output atteso (esempio):

```
main.bas  
prog1.bas
```

Nota:

- DIR e EDIR non richiedono parametri.

EFORMAT

Sintassi

EFORMAT

Descrizione

Cancella **tutti** i file nella root della SPIFFS (memoria interna).

Esempio

```
10 EFORMAT  
20 PRINT "SPIFFS pulita"
```

ELISTVARS

Sintassi:

```
ELISTVARS("file")
```

Descrizione:

ELISTVARS è equivalente a LISTVARS ma legge il file dalla **scheda SD**. Funziona come LISTVARS, ma viene usato per file salvati con ESAVEVAR.

Esempio – Visualizzare variabili salvate su SD:

```
10 ESAVEVAR "dati.json" "TEMP" 22.5  
20 ESAVEVAR "dati.json" "STATO" "OK"  
30 ELISTVARS "dati.json"
```

Output atteso:

```
Listing variables in /dati.json:  
TEMP = 22.5  
STATO = "OK"
```

ELOAD

(o LOADINT F\$)

Sintassi:

ELOAD "nomefile"
ELOAD variabile\$

Descrizione:

Il comando ELOAD carica un file .bas dalla **memoria interna SPIFFS**, indipendentemente dalla presenza di una scheda SD.

È utile per **evitare ambiguità** quando si ha sia memoria interna sia scheda SD con file omonimi.

Accetta sia:

- un **nome file** tra virgolette (es: "main.bas")
- una **variabile stringa** che contiene il nome del file (es: F\$)

☐ Il contenuto del file **sostituisce il listato BASIC in memoria.**

Esempi pratici

Esempio 1 – Caricare file da memoria interna

```
ELOAD "setup.bas"
```

Output atteso:

Il programma setup.bas viene caricato dalla SPIFFS (memoria interna).

Esempio 2 – Usare una variabile stringa

```
10 F$ = "gioco.bas"  
20 ELOAD F$  
RUN
```

Output atteso:

Il listato gioco.bas viene caricato dalla memoria interna.

ELOADVAR

Sintassi:

ELOADVAR(file chiave variabile)

Descrizione:

ELOADVAR è equivalente a LOADVAR ma legge il file da **SPIFFS**.

Funziona come LOADVAR, ma usa file salvati con ESAVEVAR.

Esempio – Caricare impostazioni dalla SPIFFS:

```
10 ELOADVAR "prefs.json" "SPEED" S
20 ELOADVAR "prefs.json" "USER" U$
```

Output atteso:

S = 150

U\$ = "Anna"

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

ERENAME

(oppure ERENAME F\$ N\$)

Sintassi

```
ERENAME "nomevecchio" "nomenuevo"  
ERENAME variabile$ variabile$
```

Descrizione

Rinomina un file presente nella **memoria interna (SPIFFS)**.
Accetta sia stringhe tra virgolette sia variabili stringa (\$).

Esempi pratici

Esempio – Alias su SPIFFS

```
ERENAME "boot.bas" "boot_old.bas"
```

Output atteso:

File renamed.

Note:

- Cerca **solo su SPIFFS** (ignora eventuale SD).
- Non modifica i contenuti: è un'operazione di metadati (rinomina).
- Utile per versionare o “archiviare” velocemente file interni.

ESAVEVAR

Sintassi:

ESAVEVAR(file chiave valore)

Descrizione:

ESAVEVAR è identico a SAVEVAR, ma salva il file **nella memoria SPIFFS** invece che sulla scheda SD.

È utile per:

- Salvare configurazioni permanenti all'interno del dispositivo
- Operare senza SD inserita

☐ I parametri sono gli stessi di SAVEVAR.

Esempio 1 – Salvare configurazione nella SPIFFS:

```
10 ESAVEVAR "prefs.json" "SPEED" 150
20 ESAVEVAR "prefs.json" "USER" "Anna"
```

Output atteso: File JSON salvato nella SPIFFS:

```
{
  "SPEED": 150,
  "USER": "Anna"
}
```

Note:

- Come SAVEVAR, ma lavora su SPIFFS
- Utile per dispositivi standalone senza SD
- Sovrascrive valori esistenti se la chiave è già presente

ESAVE

(o ESAVE F\$)

Sintassi:

ESAVE "nomefile"
ESAVE variabile\$

Descrizione:

Il comando ESAVE salva il programma BASIC attualmente in memoria nella **memoria interna (SPIFFS)**, anche se è presente una scheda SD.

È identico a SAVE, ma **forza il salvataggio su SPIFFS**, utile per conservare file fissi o di sistema senza SD.

Accetta:

- un **nome di file** tra virgolette (es. "setup.bas")
- una **variabile stringa** (es. F\$) contenente il nome

Esempi pratici

Esempio 1 – Salvataggio su SPIFFS

```
10 ESAVE "config.bas"  
RUN
```

Output atteso:

Il file config.bas viene salvato sulla memoria interna, anche con SD inserita.

Esempio 2 – Uso con variabile

```
10 F$ = "menu.bas"  
20 ESAVE F$  
RUN
```

Output atteso:

Salva menu.bas su SPIFFS.

EVERIFY

(o EVERIFY F\$)

Sintassi:

EVERIFY "nomefile"
EVERIFY variabile\$

Descrizione:

Il comando EVERIFY confronta il **programma attualmente in memoria** con un file salvato nella **memoria interna (SPIFFS)**.

Funziona esattamente come VERIFY, ma cerca **solo in SPIFFS**, ignorando eventuali file su SD.

Verifica se il listato in memoria è **identico** al contenuto del file indicato.

Esempi pratici

Esempio 1 – Verifica di file identico

```
EVERIFY "setup.bas"
```

Output atteso:

File uguale al listato in memoria

Esempio 2 – File modificato

```
10 REM VERSIONE NUOVA  
20 EVERIFY "setup.bas"  
RUN
```

Output atteso:

File diverso dal listato in memoria

Nota:

- Cerca il file **solo su SPIFFS**
- Non modifica nulla: è un controllo **non distruttivo**
- Utile per evitare **doppi salvataggi inutili**

EXAMPLES

Sintassi:

EXAMPLES

Descrizione:

Il comando EXAMPLES recupera e visualizza la lista degli esempi ufficiali disponibili nel repository GitHub di Basic32.

Per funzionare, richiede una connessione Wi-Fi attiva tramite il comando WIFI.

Esempi pratici

Esempio 1 – Visualizzare gli esempi disponibili

```
10 WIFI "ssid" "password"  
20 WAIT 2000  
30 EXAMPLES  
RUN
```

Output atteso:

Elenco degli esempi disponibili su GitHub, se la connessione è attiva.

Nota:

- Richiede una connessione Wi-Fi funzionante
- Non accetta parametri
- Gli esempi sono caricati dinamicamente da GitHub
- Utile per scoprire e caricare programmi di esempio pronti all'uso

EXP(x)

Sintassi:

EXP(x)

Descrizione:

La funzione EXP(x) restituisce il valore di **e elevato alla x**, dove **e \approx 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
```

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

Descrizione

Cancella **tutti** i file nella root della SD.

Esempio

```
10 FORMAT  
20 PRINT "SD pulita"
```

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

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 DO BLOCK 20 TO 40  
20 C = GET  
30 IF C <> -1 THEN PRINT "TASTO: "; C  
40 IF C = 81 THEN END  
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
```

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>"

Descrizione:

Il comando HTMLOBJ consente di aggiungere righe di codice HTML alla pagina web generata dal dispositivo.

Ogni riga HTML viene memorizzata in sequenza e verrà mostrata nella pagina all'avvio del server web con HTMLSTART.

Esempi pratici

Esempio 1 – Aggiungere un'intestazione HTML

```
10 HTMLOBJ "<h1>Benvenuto su Basic32</h1>"  
RUN
```

Esempio 2 – Inserire un pulsante di comando

```
10 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=PRINT CIAO\")'>Clicca qui</button>"  
RUN
```

Nota:

- Ogni riga HTML deve essere racchiusa tra virgolette
- I caratteri speciali (come doppi apici) vanno gestiti con \
- Le righe vengono accumulate fino all'esecuzione di HTMLSTART
- Supporta HTML base, pulsanti, testo, link, ecc.

HTMLSTART

Sintassi:

HTMLSTART

Descrizione:

Il comando HTMLSTART avvia il server web integrato del dispositivo, rendendo accessibile la pagina HTML definita tramite HTMLOBJ.

Il dispositivo deve essere connesso via Wi-Fi o in modalità Access Point.

Esempi pratici

Esempio completo – Controllare un LED via Web

```
10 PINMODE 2 OUTPUT NOPULL
20 WIFI "ssid" "password"
30 HTMLOBJ "<h2>Controllo LED on board</h2>"
40 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 1\")'>Accendi il LED onboard</button>"
50 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 0\")'>Spegni il LED onboard</button>"
60 HTMLSTART
RUN
```

Output atteso:

Collegandosi al dispositivo via browser, si visualizza una pagina con i pulsanti per accendere e spegnere il LED.

Nota:

- Il server web risponde su IP ottenuto da IP o IPAP
- Le azioni sono eseguite tramite `exec?cmd=...`
- Deve essere usato solo dopo WIFI o WIFIAP
- È necessario almeno un HTMLOBJ prima di HTMLSTART
- Le righe HTML vengono cancellate da MEMCLEAN(HTML)

HTTP GET

Sintassi

HTTP GET "url" TO var\$

Descrizione

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

- Se l'URL inizia con https:// viene generato errore: **HTTPS NON COMPATIBILE**.
- La variabile speciale HTTPSTATUS contiene il codice di stato (200 = OK).
- È possibile salvare gli header con HTTP HEADERS TO var\$.

Esempi pratici

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 JSON GET "query" FROM R$ TO IP$
50 PRINT "IP pubblico=";IP$
90 END
```

Esempio 2 – Meteo con 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 JSON GET "main.temp" FROM R$ TO T$
60 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 è forzato a identity: risposta sempre in testo non compresso.
- Se la risposta non è JSON, puoi stampare direttamente la stringa.

HTTP HEADERS

Sintassi

HTTP HEADERS TO var\$

Descrizione

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

- Utile per diagnostica, redirect o controllo Content-Type.
- Funziona sempre dopo un comando HTTP valido.

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 tutti gli header grezzi concatenati in un'unica stringa.
- Utile anche per controllare Content-Length o codici di redirect.

HTTP POST

Sintassi

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

Descrizione

Invia una richiesta HTTP POST (solo http://) con il contenuto e tipo specificati. La risposta viene salvata nella variabile stringa.

- HTTPS non supportato.
- Content-Type deve corrispondere al corpo inviato (application/json, application/x-www-form-urlencoded, ecc).
- Codice HTTP disponibile in HTTPSTATUS.

Esempi pratici

Esempio 1 – Echo JSON

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

Esempio 2 – Invia lettura 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 estrarre i campi con JSON GET.
- Tieni corti i body: stringhe molto lunghe possono causare instabilità.

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*

Descrizione:

Disegna un cerchio vuoto di raggio *r* nella posizione (*x*, *y*) con colore RGB.

Esempi pratici

10 ILI CIRCLE 60 120 30 255 0 0

Cerchio rosso con raggio 30 a (60, 120).

Note

- Per un cerchio pieno, usare ILI FILLCIRCLE.

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 FILLCIRCLE

Sintassi:

ILI FILLCIRCLE x y r [r g b]

Descrizione:

Disegna un **cerchio pieno** con centro in x, y, raggio r e colore RGB opzionale (di default bianco).

Esempi pratici

```
10 ILI FILLCIRCLE 100 100 30 255 0 0
```

Disegna un cerchio rosso pieno di raggio 30 centrato in (100,100).

Note:

- Se r g b non sono specificati, viene usato il colore bianco.
- È un comando immediato: il disegno resta sullo schermo finché non si fa ILI CLEAR.

ILI FILLRECT

Sintassi:

ILI FILLRECT x y larghezza altezza r g b

Descrizione:

Disegna un **rettangolo pieno** (filled rectangle) sullo schermo TFT nella posizione specificata, con la dimensione e il colore indicati.

- x, y: coordinate in pixel del punto in alto a sinistra del rettangolo.
- larghezza, altezza: dimensioni in pixel.
- [r, g, b]: colore facoltativo in RGB (0–255). Se omissso, il colore di default è **bianco (255,255,255)**.

Esempi pratici

Esempio 1 – Rettangolo bianco 50x20

```
10 ILI FILLRECT 10 10 50 20
```

Risultato: rettangolo bianco di 50x20 pixel a posizione (10,10)

Esempio 2 – Rettangolo verde

```
10 ILI FILLRECT 40 80 60 30 0 255 0
```

Risultato: rettangolo pieno verde a (40,80), largo 60 pixel e alto 30 pixel

Note

- Il comando è immediato: disegna direttamente sullo schermo.
- Usa ILI SPRITE NEW ... RECT per creare un rettangolo come sprite mobile.
- Le coordinate possono anche essere espresse con variabili.

ILI INIT (ILI9341)

Sintassi:

ILI INIT cs dc rst rot

Descrizione:

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

- cs: pin Chip Select
- dc: pin Data/Command
- rst: pin Reset
- rot: rotazione dello schermo (valori ammessi: 0, 1, 2, 3)

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).

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).

ILI INIT TOUCH (XPT2046)

Sintassi

ILI INIT TOUCH <cs> <rotation>

Descrizione

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

- <cs> è il pin **Chip Select** del controller touch.
- <rotation> definisce l'orientamento del display (0–3), e viene sincronizzato con la rotazione del TFT.
- Dopo l'inizializzazione, il touch può essere usato con i comandi:
 - ILI TOUCH CALIBRATE
 - ILI TOUCH SPRITE ...
 - ILI TOUCH AREA ...

Se eseguito più volte, 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.

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.
- 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

Descrizione:

Disegna una linea tra due punti sullo schermo. Può essere opzionalmente colorata.

- x1, y1: coordinate del punto iniziale
- x2, y2: coordinate del punto finale
- r, g, b: (opzionale) colore RGB della linea (valori da 0 a 255)

Se il colore non è specificato, viene usato **bianco** come default.

Esempi pratici

Esempio 1 – Linea bianca

```
10 ILI INIT 17 16 5 1
20 ILI LINE 10 10 100 50
```

Risultato: viene disegnata una linea bianca da (10,10) a (100,50)

Esempio 2 – Linea rossa

```
10 ILI INIT 17 16 5 1
20 ILI LINE 20 60 120 60 255 0 0
```

Risultato: una linea rossa orizzontale

Note

- Le coordinate devono essere **all'interno dei limiti** dello schermo TFT (tipicamente 320x240).
- Il colore può essere espresso come valori RGB, ognuno da 0 a 255.
- Se si ridisegna sullo stesso punto, la linea sovrascrive ciò che c'era.

ILI PIXEL

Sintassi:

ILI PIXEL x y r g b

Descrizione:

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

- x, y: coordinate del pixel
- r, g, b: (opzionale) colore del pixel (default: bianco)

Esempi pratici

Esempio 1 – Pixel bianco

```
10 ILI INIT 17 16 5 1
20 ILI PIXEL 50 50
```

Risultato: un piccolo punto bianco compare in (50,50)

Esempio 2 – Pixel blu

```
10 ILI INIT 17 16 5 1
20 ILI PIXEL 100 100 0 0 255
```

Risultato: pixel blu in (100,100)

Note

- Molto utile per test grafici, effetti pixel-art o per disegnare manualmente bitmap.
- Se le coordinate sono fuori dallo schermo, il comando non ha effetto.
- Il colore può essere regolato dinamicamente per creare animazioni o effetti.

ILI RECT

Sintassi:

ILI RECT x y w h r g b

Descrizione:

Disegna un rettangolo vuoto (solo il bordo) alle coordinate specificate, con larghezza e altezza date.

- x y: coordinate dell'angolo superiore sinistro
- w h: larghezza e altezza del rettangolo
- r g b: (opzionale) colore del bordo, default bianco

Esempi pratici

Esempio 1 – Rettangolo bianco

```
10 ILI INIT 17 16 5 1
20 ILI RECT 20 20 100 50
```

Risultato: rettangolo bianco vuoto 100x50 in (20,20)

Esempio 2 – Rettangolo rosso

```
10 ILI INIT 17 16 5 1
20 ILI RECT 30 40 60 30 255 0 0
```

Risultato: rettangolo rosso vuoto in (30,40) di dimensione 60x30

Note

- Solo il bordo viene disegnato. Se vuoi un rettangolo pieno, usa ILI FILLRECT.
- Usalo per interfacce grafiche, finestre, pulsanti, ecc.
- Il colore è opzionale: se omesso, è bianco.

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 SPRITE CHAR

Sintassi:

ILI SPRITE NEW id CHAR x y "C" size r g b

Descrizione:

Crea uno sprite che visualizza un carattere singolo alla posizione indicata, con dimensione e colore specificati.

- id: identificatore univoco dello sprite (da 0 a MAX_SPRITES)
- "C": carattere tra virgolette (es. "A")
- x, y: coordinate del carattere
- size: dimensione del testo (1 = normale, 2 = doppio, ecc.)
- r, g, b: colore del carattere (0–255)

Esempi pratici

Esempio 1 – Carattere 'A' in verde

```
10 ILI SPRITE NEW 1 CHAR 20 100 "A" 2 0 255 0
20 ILI SPRITE DRAW
```

Visualizza il carattere A in verde, dimensione 2, a (20,100)

Modifica successiva con SETCHAR

```
ILI SPRITE SETCHAR 1 "B" 255 0 0
```

Cambia il carattere in **B** e lo rende rosso.

Note

- Ogni CHAR può essere modificato usando ILI SPRITE SETCHAR.
- Usare virgolette per indicare il carattere ("X"), oppure codice ASCII.
- Funziona solo se lo sprite è stato inizializzato come CHAR.

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

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

```
10 ILI SPRITE NEW 0 RECT 10 10 40 40 255 0 0
20 ILI SPRITE DRAW
```

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 FRAME

(Incluso in ILI SPRITE NEW con tipo "FRAME")

Sintassi:

ILI SPRITE NEW id FRAME x y w h r g b

Descrizione:

Disegna un rettangolo vuoto (solo bordo), come una cornice.

Esempi pratici

```
10 ILI SPRITE NEW 1 FRAME 50 50 80 40 0 255 0
20 ILI SPRITE DRAW
```

Disegna una cornice verde.

Note

- Usa ILI SPRITE NEW con tipo "FRAME" per creare.
- È simile a RECT, ma senza riempimento.

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 LINE

Sintassi:

ILI SPRITE NEW id LINE x y x2 y2 r g b

Descrizione:

Disegna una linea tra due punti specificati con un colore RGB.

Esempi pratici

```
10 ILI SPRITE NEW 1 LINE 10 60 120 60 255 255 0
20 ILI SPRITE DRAW
```

Disegna una linea gialla da sinistra a destra.

Note

- x y sono le coordinate iniziali.
- x2 y2 sono le coordinate finali.
- Non è necessario usare ILI SPRITE MOVE per ridisegnare: è meglio usare ILI SPRITE NEW o ILI SPRITE SET... per aggiornare le proprietà.

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:

ILI SPRITE NEW id tipo x y dimensione/parametri ["testo"/valore] [r g b]

Descrizione:

Crea uno sprite da disegnare sul display ILI9341.

Gli sprite possono essere di diversi tipi: RECT, FRAME, CIRCLE, LINE, CHAR, NUM, TEXT, PIXELS.

Esempi pratici

Rettangolo pieno (RECT)

```
10 ILI SPRITE NEW 0 RECT 20 20 60 40 255 0 0
```

Disegna un rettangolo rosso di 60x40 a (20,20)

Cornice (FRAME)

```
20 ILI SPRITE NEW 1 FRAME 100 20 60 40 0 255 0
```

Disegna una cornice verde 60x40 a (100,20)

Cerchio (CIRCLE)

```
30 ILI SPRITE NEW 2 CIRCLE 60 120 30 0 0 255
```

Disegna un cerchio blu raggio 30 a (60,120)

Linea (LINE)

```
40 ILI SPRITE NEW 3 LINE 20 180 120 200 255 255 0
```

Disegna una linea gialla da (20,180) a (120,200)

Carattere singolo (CHAR)

```
50 ILI SPRITE NEW 4 CHAR 160 40 "A" 3 0 255 255
```

Disegna la lettera "A" in ciano, grandezza 3, a (160,40)

Numero (NUM)

```
60 N = 123
```

```
70 ILI SPRITE NEW 5 NUM 160 80 N 2
```

Disegna il numero 123 in bianco, grandezza 2, a (160,80)

Testo (TEXT)

```
80 A$ = "WORLD"
```

```
90 ILI SPRITE NEW 6 TEXT 20 240 2 "HELLO " + A$ 255 0 255
```

Disegna il testo HELLO WORLD in magenta, grandezza 2, a (20,240)

Pixels (PIXELS) *(solo se hai routine bitmap)*

```
100 ILI SPRITE NEW 7 PIXELS 40 40 16 16 MyBitmapData
```

Disegna un'immagine 16x16 a (40,40) usando i dati MyBitmapData

Renderizza gli sprite creati

```
110 ILI SPRITE DRAW
```

Note

- I parametri variano in base al tipo:
 - RECT, FRAME: larghezza, altezza
 - CIRCLE: raggio
 - LINE: x2, y2 (coordinate finali)
 - CHAR: carattere (ASCII o lettera tra virgolette), dimensione
 - NUM: valore, dimensione
 - TEXT: dimensione, espressione testuale (stringa, variabile, concatenazione, funzione)
 - PIXELS: dati bitmap
- Il colore RGB è opzionale (default: bianco).
- Supporta variabili numeriche e di testo a seconda del tipo.
- Dopo la creazione, disegna gli sprite con ILI SPRITE DRAW.

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 (\$).
-

Esempi pratici (copiabili)

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
65 ILI SPRITE DRAW
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
65 ILI SPRITE DRAW
66 ILI 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 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
65 ILI SPRITE DRAW
66 ILI 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** (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 = $\min(\text{larghezza}, \text{altezza})/2$.
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 0
45 ILI SPRITE NEW 2 CIRCLE 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
65 ILI SPRITE DRAW
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 0
45 ILI SPRITE NEW 2 CIRCLE 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
65 ILI SPRITE DRAW
66 ILI 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 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 0  
41 ILI SPRITE NEW 2 CIRCLE 120 100 10 10 0 255 0  
42 ILI SPRITE NEW 3 CIRCLE 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  
65 ILI SPRITE DRAW  
66 ILI SPRITE COLLISIONC 1 TO ANY SET HIT  
67 IF HIT THEN PRINT "1 ha toccato almeno uno (circolare)"  
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]

Descrizione:

Visualizza un testo sul display ILI9341 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 automaticamente in stringa)

I parametri [r g b] sono opzionali e definiscono il colore del testo (valori 0–255). Se omessi, il colore è bianco.

Esempi pratici

Esempio 1 – Testo fisso

```
10 ILI TEXT 10 50 2 "HELLO ILI" 0 255 255
```

Scrive HELLO ILI in ciano, grandezza 2, a posizione (10, 50).

Esempio 2 – Testo concatenato con variabile stringa

```
10 A$ = "WORLD"  
20 ILI TEXT 20 80 2 "HELLO " + A$
```

Esempio 3 – Concatenazione con numero

```
10 N = 123  
20 ILI TEXT 10 120 2 "VALUE=" + N 255 0 0
```

Scrive VALUE=123 in rosso.

Note

- Non è più obbligatorio racchiudere tutto tra virgolette: ora puoi concatenare testo, variabili e numeri con +.
- I valori r g b sono opzionali (0–255).
- L'ultima scritta resta visibile finché non viene cancellata con ILI CLEAR.
- Supporta variabili ed espressioni stringa/numeriche.

ILI TOUCH AREA

Sintassi

ILI TOUCH AREA x y w h GOTO line
ILI TOUCH AREA x y w h SET var

Descrizione

Verifica se il tocco cade dentro l'area rettangolare specificata (coordinate schermo ILI, **dopo** calibrazione).

- Con **GOTO** salta a line quando l'area viene toccata.
- Con **SET** scrive **1** se toccata, **0** altrimenti in var (numerica o stringa \$).
- Accetta **dimensioni negative** (w, h): l'area 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
70 GOTO 60
200 PRINT "Tap nell'area!"
210 END
```

Area con variabile

```
10 ILI TOUCH AREA 20 20 100 60 SET T
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** del display e **calibrazione** (minRawX/maxRawX/minRawY/maxRawY).

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
ILI TOUCH SPRITE id SET var
ILI TOUCH SPRITE ANY SET var

Descrizione

Rileva il tocco **sul bounding box** di uno sprite:

- id GOTO line (già esistente): se tocchi lo sprite con ID id, salta a line.
- id SET var (nuovo): mette **1/0** in var se il tocco cade dentro lo sprite id.
- ANY SET var (nuovo): mette in var **l'ID del primo sprite toccato**, oppure **0** se nessuno.

Funziona con tutti i tipi di sprite (RECT, FRAME, CIRCLE, LINE, TEXT, CHAR, NUM, PIXELS...), purché w/h rappresentino correttamente l'area occupata.

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
40 IF HIT THEN PRINT "Hai toccato lo sprite 5"
50 GOTO 30
```

Qualunque sprite → variabile = ID

```
10 ILI TOUCH SPRITE ANY SET S
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
40 GOTO 30
200 PRINT "Toccato lo sprite 3!"
210 END
```

Note

- Gli sprite **devono essere attivi e visibili** (altrimenti il tocco viene ignorato).
- Se il touch non è disponibile o non è premuto, **non avviene alcun salto** e le variabili con SET assumono **0**.

- Il test usa il **bounding box** (x,y,w,h). Per sprite circolari, se vuoi un test “circolare” reale, puoi mantenere COLLISIONC per le collisioni; per il touch il bbox è più che sufficiente e più veloce.

INITRTC

Sintassi:

INITRTC

Descrizione:

Il comando INITRTC inizializza il modulo RTC (Real Time Clock) collegato al dispositivo. Deve essere eseguito prima di utilizzare comandi come TIMEH, TIMEM, TIMES, DATEY, DATEM, DATED.

Serve a sincronizzare il sistema con l'orario e la data correnti forniti dal modulo RTC esterno (tipicamente DS3231).

Esempi pratici

Esempio 1 – Inizializzare l'RTC e stampare l'ora

```
10 INITRTC
20 PRINT TIMEH; ":"; TIMEM; ":"; TIMES
RUN
```

Output atteso:

14:03:52

Nota:

- Deve essere eseguito prima di leggere l'ora/data dal modulo RTC
- Il modulo deve essere collegato correttamente via I2C
- Compatibile con moduli DS1307 / DS3231
- Funziona solo se l'RTC è presente e alimentato

INITSD

Sintassi:

INITSD <cs> <mosi> <miso> <sck>

Descrizione:

Il comando INITSD inizializza la scheda SD specificando i pin da usare:

- cs → Chip Select
- mosi, miso, sck → linee SPI

È necessario eseguirlo prima di usare comandi che leggono o scrivono file su SD (es. LOAD, SDFREE).

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 con pin personalizzati e altri device SPI come display,touch ecc...

```
10 PINMODE 17 OUTPUT NOPULL      'pin cs del display ili
20 DWRITE 17 1                   ' metti a HIGH il pin
30 PINMODE 4 OUTPUT NOPULL       ' pin cs del touch ili
40 DWRITE 4 1                     ' metti a HIGH il pin
50 INITSD 13 23 19 18
60 PRINT SDFREE
RUN
```

Output atteso:

Mostra lo spazio libero sulla SD se inizializzata correttamente.

Note:

- Inizializza la scheda SD con SPI software o hardware
- Serve una scheda SD formattata FAT32
- La corretta assegnazione dei pin dipende dal cablaggio
- Dopo l'inizializzazione, la SD è pronta per letture/scritture
- Controllare SDFREE per verificare il montaggio

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.

- $\text{INT}(4.7) \rightarrow 4$
- $\text{INT}(-2.3) \rightarrow -2$

□ 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(1) * 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

JSON GET

Sintassi

JSON GET path\$ FROM src\$ TO var\$

Descrizione

Estrae un valore da una stringa JSON.

- path\$ può contenere campi annidati (main.temp) e indici array (weather[0].description).
- Se il percorso non esiste → errore: **JSON Path not found**.

Esempi pratici

Esempio 1 – Meteo da 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 JSON GET "main.temp" FROM R$ TO T$
50 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 JSON GET "headers.Host" FROM R$ TO H$
40 PRINT "Server visto come host:";H$
90 END
```

Note

- Se la stringa non inizia con {, probabilmente non è JSON valido.
- Gli indici partono da 0.
- Sempre meglio controllare HTTPSTATUS=200 prima di estrarre.

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 – 2x2

```
10 KPINIT 2 2 13 12 14 27
20 PRINT "KEYPAD PRONTO"
RUN
```

Output atteso:

KEYPAD PRONTO

Esempio 2 – 3x4 classico

```
10 KPINIT 4 3 2 3 4 5 6 7 8
RUN
```

Esempio 3 – 4x4

```
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 4x4

```
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].

LEDCSTATUS

Sintassi:

LEDCSTATUS

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 LEDCSTATUS
```

→ 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 LEDCSTATUS
```

→ Verifica che gli ID 0 e 1 siano “ATTACHED” con PIN e range.

Output atteso (esempio indicativo):

```
=== LEDC / 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).

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 – Sottstringa 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 ($A(1) = \dots$) se non prima di un DIM
- Puoi usare LET per migliorare la leggibilità, anche se non è obbligatorio

LIST

Sintassi:

LIST

Descrizione:

Il comando LIST mostra sul terminale **tutte le linee di programma attualmente in memoria**, ordinate per numero di riga.

È utile per:

- **visualizzare** il codice scritto
- **modificare manualmente** una riga esistente (riscrivendola)
- **verificare** il contenuto prima di eseguire

Non prende parametri.

Il listato mostrato è quello **attualmente caricato in RAM**, non da file.

Esempi pratici

Esempio – Visualizzare un programma scritto

```
10 PRINT "CIAO"  
20 END
```

LIST

Output atteso:

```
10 PRINT "CIAO"  
20 END
```

Nota:

- Per cancellare tutto il listato dalla memoria usa NEW
- Le righe possono essere riscritte digitando di nuovo il numero riga

LISTVARS

Sintassi:

```
LISTVARS("file")
```

Descrizione:

LISTVARS legge tutte le variabili contenute in un file JSON dalla memoria interna **SPIFFS**. Mostra a video ogni coppia **chiave = valore** salvata con il comando SAVEVAR. È utile per verificare il contenuto di configurazioni o dati locali.

Esempio – Visualizzare variabili salvate in SPIFFS:

```
10 SAVEVAR "config.json" "NOME" "LUCA"  
20 SAVEVAR "config.json" "LIVELLO" 7  
30 SAVEVAR "config.json" "ATTIVO" TRUE  
40 LISTVARS "config.json"
```

Output atteso:

```
Listing variables in /config.json:  
NOME = "LUCA"  
LIVELLO = 7  
ATTIVO = true
```

LOAD

(o LOAD F\$)

Sintassi:

LOAD "nomefile"
LOAD variabile\$

Descrizione:

Il comando LOAD carica un file .bas dalla memoria attiva **SD** e lo trasferisce nella **memoria programma**, sovrascrivendo il listato attuale.

Accetta sia:

- un **nome di file diretto** tra virgolette (es: "test.bas")
- una **variabile stringa** che contiene il nome del file (es: F\$)

Se è presente una scheda SD, LOAD legge da lì; altrimenti, dalla memoria interna.

☐ L'uso di LOAD **sostituisce completamente** il programma in memoria.

Esempi pratici

Esempio 1 – Caricare un file da SD (se presente)

```
LOAD "menu.bas"
```

Output atteso:

Il listato presente in menu.bas viene caricato nella memoria.

Esempio 2 – Caricare da variabile

```
10 LET F$ = "config.bas"  
20 LOAD F$  
RUN
```

Output atteso:

Carica il programma dal file config.bas.

Esempio 3 – Errore se il file non esiste

→ Se il file specificato non esiste, viene mostrato un errore (es: File not found).

LOADGIT

Sintassi:

LOADGIT "<nomefile>"

Descrizione:

Il comando LOADGIT scarica e carica automaticamente un file di esempio presente nel repository GitHub ufficiale di Basic32.

Il nome del file deve corrispondere a uno degli esempi elencati con il comando EXAMPLES. Richiede una connessione Wi-Fi attiva.

Esempi pratici

Esempio 1 – Caricare un esempio chiamato blink.bas

```
10 WIFI "ssid" "password123"  
20 LOADGIT "blink.bas"  
RUN
```

Output atteso:

Carica il contenuto del file blink.bas direttamente da GitHub nella memoria del programma.

Nota:

- Il nome del file deve essere esatto e racchiuso tra virgolette
- I file vengono caricati dalla repository ufficiale Basic32 su GitHub
- Sovrascrive il programma attuale in memoria
- Richiede una rete Wi-Fi funzionante (WIFI)
- Funziona bene insieme al comando EXAMPLES

LOADVAR

Sintassi:

LOADVAR(file chiave variabile)

Descrizione:

Il comando LOADVAR carica un valore da un file JSON su **SD** e lo assegna a una variabile BASIC.

È usato per:

- Caricare valori salvati con SAVEVAR
- Inizializzare parametri utente
- Ripristinare stato al riavvio

☐ La variabile può essere:

- numerica (X)
 - stringa (X\$)
-

Esempio 1 – Caricare valori salvati:

```
10 LOADVAR "config.json" "NOME" A$
20 LOADVAR "config.json" "LIVELLO" A
30 PRINT A$, A
```

Output atteso:

```
A$ = "Mario"
A = 42
```

Note:

- Se la chiave non esiste, viene mostrato un errore
- Il file deve essere valido JSON
- Per SPIFFS vedi ELOADVAR

LISTTIMEZONES

Sintassi:

LISTTIMEZONES

Descrizione:

Il comando **LISTTIMEZONES** mostra l'elenco completo dei fusi orari disponibili, con i relativi nomi e offset rispetto a UTC.

È utile per sapere quale valore usare con il comando TIMEZONE.

Esempi pratici

Esempio 1 – Visualizzare i fusi orari disponibili

```
10 LISTTIMEZONES
```

```
RUN
```

Output atteso (estratto):

lista timezone

Nota:

- Gli offset sono da usare direttamente con il comando TIMEZONE
- Non imposta nulla: è un comando informativo
- Utile per scegliere il fuso corretto senza errori

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.

- $\text{LOG}(1) = 0$
- $\text{LOG}(e) = 1$
- $\text{LOG}(x)$ 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 $f(x) = \text{LOG}(x) * x$

```
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:

$$\text{LOG}_{10}(X) = \text{LOG}(X) / \text{LOG}(10)$$

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.

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 – Iscrivere 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 topic 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:

Il comando NOWCLR cancella il contenuto della variabile speciale NOWRECV\$, che contiene l'ultimo messaggio ricevuto tramite ESP-NOW.

È utile per **svuotare il buffer di ricezione** e prevenire la ripetizione involontaria di elaborazioni basate su vecchi messaggi.

Questo comando **non accetta parametri** e deve essere usato **dopo aver processato un messaggio ricevuto**, ad esempio dopo un PRINT, un LET, o un controllo IF.

Esempi pratici

Esempio 1 – Ricezione semplice con pulizia

```
10 NOWINIT
20 IF NOWRECV$ <> "" THEN PRINT "Ricevuto: "; NOWRECV$
30 IF NOWRECV$ <> "" THEN NOWCLR
```

→ Stampa il messaggio ricevuto e poi lo cancella.

Esempio 2 – Ricezione in loop

```
10 NOWINIT
20 PRINT "In ascolto..."
30 GOTO 100
100 IF NOWRECV$ <> "" THEN LET M$ = NOWRECV$
110 IF NOWRECV$ <> "" THEN PRINT "Messaggio: "; M$
120 IF NOWRECV$ <> "" THEN NOWCLR
130 GOTO 100
```

→ Loop continuo che legge, stampa e cancella i messaggi in arrivo.

Output atteso

Quando viene ricevuto un messaggio:

Messaggio: CIAO

Se nessun messaggio è presente, non succede nulla. Il buffer non viene cancellato finché NOWCLR non viene eseguito.

Note

- NOWCLR agisce **solo** sulla variabile NOWRECV\$
- Deve essere usato per evitare che un messaggio venga letto più volte
- Non è necessario se si sovrascrive NOWRECV\$ con LET, ma è consigliato per chiarezza
- L'uso corretto di NOWCLR assicura che ogni messaggio venga elaborato una sola volta

NOWINIT (ESP-NOW)

Sintassi:

NOWINIT

Descrizione:

Il comando NOWINIT inizializza la comunicazione ESP-NOW, attivando la modalità stazione Wi-Fi (WIFI_STA) e predisponendo il dispositivo per l'invio e la ricezione di messaggi tramite protocollo **ESP-NOW**.

Una volta eseguito correttamente, viene anche aggiornata la variabile speciale NOWMAC\$, che contiene il **MAC address locale** del dispositivo, utile per identificare il mittente o configurare il destinatario da parte di un altro dispositivo.

Se la procedura di inizializzazione fallisce, viene restituito un errore:

ESP-NOW initialization error.

Esempi pratici

Esempio 1 – Inizializzare ESP-NOW e mostrare il MAC address

```
10 NOWINIT
20 PRINT "MAC locale: "; NOWMAC$
```

→ Inizializza ESP-NOW e stampa il MAC del dispositivo.

Esempio 2 – Eseguire NOWINIT prima dell'invio o della ricezione

```
10 NOWINIT
20 PRINT "ESP-NOW pronto"
```

→ Prepara il dispositivo per usare NOWSEND e leggere NOWRECV\$.

Output atteso

Se tutto funziona:

MAC locale: 14:33:5C:0E:9A:B8

Se c'è un errore:

ESP-NOW initialization error.

Note

- Deve essere eseguito **prima** di ogni utilizzo di NOWSEND, NOWCLR, NOWRCV\$
- Imposta automaticamente la modalità Wi-Fi in WIFI_STA
- Aggiorna la variabile di sistema NOWMAC\$
- L'inizializzazione è necessaria anche per ricevere messaggi
- Il comando **non richiede parametri**

NOWSEND

Sintassi:

NOWSEND "MAC" "messaggio"

Descrizione:

Il comando NOWSEND consente di inviare un messaggio stringa a un altro dispositivo tramite protocollo **ESP-NOW**, specificando il **MAC address del destinatario** e il contenuto del messaggio.

Il MAC address deve essere nel formato "XX:XX:XX:XX:XX:XX" e racchiuso tra virgolette. Il messaggio deve anch'esso essere una stringa, sempre tra virgolette.

Questo comando richiede che NOWINIT sia stato eseguito **prima**, per inizializzare il sistema ESP-NOW.

Se il MAC non è valido, o il formato è errato, viene generato un errore di sintassi:

```
?SYNTAX ERROR  
Invalid MAC in NOWSEND
```

Esempi pratici

Esempio 1 – Inviare un messaggio

```
10 NOWINIT  
20 PRINT "MAC: "; NOWMAC$  
30 NOWSEND "24:62:AB:F2:4D:CC" "CIAO"  
40 NOWCLR
```

→ Invia "CIAO" al dispositivo con MAC 24:62:AB:F2:4D:CC.

Esempio 2 – Inviare una variabile come messaggio

```
10 NOWINIT  
20 LET M$ = "OK"  
30 NOWSEND "24:62:AB:F2:4D:CC" M$  
40 NOWCLR
```

→ Invia il contenuto della variabile M\$ come messaggio.

Output atteso

Se l'invio va a buon fine, non viene mostrato nulla di particolare (a meno che non venga gestito manualmente).

Se il MAC è errato:

```
?SYNTAX ERROR  
Invalid MAC in NOWSEND
```

Note

- Il comando NOWINIT **deve essere eseguito prima** di NOWSEND
- Il MAC deve essere specificato in formato esadecimale, separato da :, es.
"24:62:AB:F2:4D:CC"
- Il messaggio può essere una **stringa fissa** o una **variabile stringa**
- Non è possibile inviare messaggi a più destinatari contemporaneamente
- Se il destinatario non è raggiungibile o non ha eseguito NOWINIT, il messaggio può andare perso

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 CRCLLEN 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).
- CRCLLEN → 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.
- CRCLLEN 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:

Apri 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>

Descrizione:

Disegna un cerchio sul display OLED, centrato in <x>, <y> e con raggio <raggio>. Il cerchio è tracciato in bianco sullo sfondo nero.

Esempi pratici

Esempio 1 – Cerchio al centro dello schermo:

```
10 OLED INIT 128 32 3C
20 OLED CLEAR
30 OLED CIRCLE 64 16 10
```

Esempio 2 – Animazione con raggio crescente:

```
10 OLED INIT 128 32 3C
20 FOR R = 1 TO 16
30 OLED CLEAR
40 OLED CIRCLE 64 16 R
50 WAIT 50
60 NEXT R
```

Note:

- Il disegno avviene immediatamente. Se usi SPRITE DRAW, assicurati che non cancelli quanto disegnato.
- Per disegnare un cerchio riempito, si può estendere con un futuro comando FILLED CIRCLE.

OLEDDATA / ENDOLEDDATA

Sintassi:

```
OLEDDATA nome  
DATA val1, val2, ..., valN  
...  
ENDOLEDDATA
```

Descrizione:

OLEDDATA definisce un blocco di dati bitmap che può essere successivamente utilizzato per disegnare sprite (immagini pixel per pixel) su un display OLED.

Ogni riga DATA rappresenta **una riga dell'immagine**. Ogni valore 1 o 0 corrisponde a un singolo pixel **bianco (1)** o **spento/nero (0)**.

L'immagine è quindi una matrice binaria, da sinistra a destra, riga per riga.

Il nome specificato dopo OLEDDATA è **case-insensitive** e verrà usato in altri comandi (es. OLED SPRITE DATA) per richiamare questa struttura.

Le righe DATA **devono essere consecutive** e comprese tra OLEDDATA e ENDOLEDDATA. La larghezza viene calcolata automaticamente dalla prima riga DATA. Tutte le righe successive devono avere lo **stesso numero di valori**.

Esempi pratici

Esempio 1 – Disegno di un cuoricino (9x5)

```
10 OLEDDATA HEART  
20 DATA 0,1,0,0,0,0,0,1,0  
30 DATA 1,1,1,0,0,1,1,1,1  
40 DATA 1,1,1,1,1,1,1,1,1  
50 DATA 0,1,1,1,1,1,1,1,0  
60 DATA 0,0,1,1,1,1,1,0,0  
70 DATA 0,0,0,1,1,1,0,0,0  
80 DATA 0,0,0,0,1,0,0,0,0  
90 ENDOLEDDATA  
100 OLED INIT 128 32 3C  
110 OLED CLEAR  
120 OLED SPRITE DATA 0 HEART 10 10
```

Note:

- I valori DATA devono essere 0 o 1
- Il numero di elementi per riga determina la **larghezza** dell'immagine
- Il numero di righe DATA determina l'**altezza**

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, ma azzerà ciò che è visivamente mostrato in quel momento.

Esempi pratici

Esempio 1 – Pulizia schermo:

```
10 OLED INIT 128 32 3C
20 OLED CLEAR
```

Esempio 2 – Pulizia ad ogni ciclo di animazione:

```
10 OLED INIT 128 32 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 (che vanno ridisegnati con OLED SPRITE DRAW).

OLED FILLRECT

Sintassi:

OLED FILLRECT <x> <y> <larghezza> <altezza> [aggiorna] [colore]

Descrizione:

Disegna un rettangolo riempito in posizione <x>, <y> con le dimensioni indicate. Può essere utile per evidenziare aree o cancellare porzioni con il colore nero.

Esempi pratici

Esempio 1 – Quadrato bianco centrato:

```
10 OLED FILLRECT 54 12 20 8
```

Esempio 2 – Riempimento nero senza aggiornamento:

```
10 OLED FILLRECT 0 0 128 32 0 BLACK  
20 OLED UPDATE
```

Note:

- Il colore BLACK è utile per cancellare aree.
- L'aggiornamento può essere posticipato per prestazioni migliori.

OLED INIT (DISPLAY SSD1306)

Sintassi:

OLED INIT <larghezza> <altezza> <indirizzo_hex> [sda scl]

Descrizione:

Inizializza il display OLED specificando dimensioni e indirizzo I2C.

- <larghezza>: in pixel (es. 128)
- <altezza>: in pixel (es. 32 o 64)
- <indirizzo_hex>: indirizzo I2C in **esadecimale** (senza 0x, es. 3C)

☐ Deve essere eseguito **prima di qualsiasi altro comando OLED**.

Esempi pratici

10 OLED INIT 128 32 3C

oppure

10 OLED INIT 128 64 3C

Note:

- L'indirizzo tipico è 3C (per SSD1306 I2C).
- Dopo l'inizializzazione, il flag `oled_enabled` viene attivato e i comandi OLED diventano utilizzabili.
- Se non vengono specificati nel comando i pin `sda` e `scl` verranno usati quelli di default. `Sda` pin 21 `scl` pin 22

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 (comportamento standard).
-

Esempi pratici

```
10 OLED INIT 128 32 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> [aggiorna] [colore]

Descrizione:

Disegna una linea da <x1>, <y1> a <x2>, <y2> sul display OLED.
Il colore può essere WHITE (default) o BLACK.

Esempi pratici

Esempio 1 – Diagonale su schermo:

```
10 OLED INIT 128 32 3C
20 OLED LINE 0 0 127 31
```

Esempio 2 – Linea orizzontale nera senza aggiornare:

```
10 OLED LINE 0 15 127 15 0 BLACK
20 OLED UPDATE
```

Note:

- Parametri aggiorna e colore sono opzionali.
- Usare OLED UPDATE per ottimizzare più disegni insieme.

OLED PIXEL

Sintassi:

OLED PIXEL <x> <y> [aggiorna] [colore]

Descrizione:

Disegna un singolo pixel sul display OLED nelle coordinate <x>, <y>.

Il pixel è bianco per impostazione predefinita, ma può essere disegnato anche in nero.

Il parametro aggiorna specifica se aggiornare subito lo schermo (1) o rimandare con OLED UPDATE.

Esempi pratici

Esempio 1 – Pixel in alto a sinistra:

```
10 OLED INIT 128 32 3C
20 OLED CLEAR
30 OLED PIXEL 0 0
```

Esempio 2 – Pixel nero senza aggiornamento:

```
10 OLED INIT 128 32 3C
20 OLED PIXEL 10 10 0 BLACK
30 OLED UPDATE
```

Note:

- Se non viene specificato aggiorna, il valore predefinito è 1.
- Colore predefinito: **bianco**.
- Il comando OLED UPDATE forza l'aggiornamento manuale.

OLED RECT

Sintassi:

OLED RECT <x> <y> <larghezza> <altezza> [aggiorna] [colore]

Descrizione:

Disegna un rettangolo vuoto con l'angolo superiore sinistro in <x>, <y>, dimensioni date da larghezza e altezza.

Esempi pratici

Esempio 1 – Bordo rettangolare intorno allo schermo:

```
10 OLED RECT 0 0 128 32
```

Esempio 2 – Rettangolo nero che cancella area:

```
10 OLED RECT 30 10 20 10 1 BLACK
```

Note:

- Per riempire un rettangolo, usa OLED FILLRECT.

OLED UPDATE

Sintassi:

OLED UPDATE

Descrizione:

Aggiorna il display OLED con tutte le operazioni grafiche precedentemente eseguite. È utile quando si usano comandi con aggiornamento disabilitato (aggiorna = 0), per disegnare tutto in un colpo solo e **evitare l'effetto animazione pixel-per-pixel**.

Esempi pratici

Esempio 1 – Disegno in blocco di una figura:

```
10 OLED INIT 128 32 3C
20 OLED CLEAR
30 FOR I = 0 TO 10
40 OLED PIXEL I I 0
50 NEXT I
60 OLED UPDATE
```

Esempio 2 – Più oggetti disegnati insieme:

```
10 OLED FILLRECT 0 0 30 10 0
20 OLED LINE 0 0 127 31 0
30 OLED TEXT 32 10 1 "PRONTO" 0
40 OLED UPDATE
```

Note:

- Utilissimo per migliorare la fluidità e ridurre il flickering.
- Non ha parametri: esegue il display() sulla memoria video.
- Se tutti i comandi grafici usano aggiorna = 1, OLED UPDATE non è necessario.

OLED TEXT

Sintassi:

OLED TEXT <x> <y> <dimensione> <testo/expr> [aggiorna]

Descrizione:

Stampa il testo specificato nella posizione indicata, con la dimensione data.

Il parametro <testo/expr> può essere:

- una stringa letterale tra virgolette
- una variabile stringa (A\$)
- un'espressione concatenata con + (es. "VAL=" + N, "Ciao " + A\$ + "!")
- una funzione stringa (es. LEFT\$(A\$,3), STR\$(X))
- un numero o un'espressione numerica (automaticamente convertita in stringa)

L'aggiornamento ([aggiorna]) è opzionale: se omissso, serve chiamare OLED UPDATE per visualizzare.

Esempi pratici

Esempio 1 – Titolo al centro

```
10 OLED TEXT 32 12 2 "HELLO"
```

Esempio 2 – Scritta senza aggiornare subito

```
10 OLED TEXT 10 10 1 "Bye" 0  
20 OLED UPDATE
```

Esempio 3 – Concatenazione con variabile numerica

```
10 T = 22  
20 OLED TEXT 0 0 1 "TEMP=" + T + " C"
```

Esempio 4 – Variabile stringa

```
10 A$ = "WORLD"  
20 OLED TEXT 0 16 1 "HELLO " + A$
```

Note

- La dimensione del testo influisce su altezza e larghezza.
- OLED TEXT può essere usato più volte senza aggiornare per performance migliori.
- Ora è possibile concatenare testo, variabili e numeri con +.
- Se il parametro non è stringa, viene convertito automaticamente.

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 = $\min(\text{larghezza}, \text{altezza})/2$.
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 DATA

Sintassi:

OLED SPRITE DATA id nome x y

Descrizione:

OLED SPRITE DATA disegna sul display uno sprite definito precedentemente tramite OLEDDATA.

id è un numero identificativo (intero) dello sprite, utile per spostarlo o gestirlo con altri comandi (MOVE, DELETE, HIDE, ecc.).

nome è il nome definito nel blocco OLEDDATA.

x, y sono le coordinate sul display in cui iniziare a disegnare l'immagine.

Il sistema calcola automaticamente la **larghezza** e **altezza** dell'immagine leggendo il blocco dati associato al nome.

Esempi pratici

Esempio – Visualizzare lo sprite

```
10 OLEDDATA SQUARE
20 DATA 1,1,1
30 DATA 1,0,1
40 DATA 1,1,1
50 ENDOLEDDATA
60 OLED INIT 128 32 3C
70 OLED CLEAR
80 OLED SPRITE DATA 1 SQUARE 10 10
```

Output: Un quadrato bianco disegnato a posizione (10,10)

Note:

- L'id è obbligatorio per identificare lo sprite e modificarlo
- Se riposizionato con OLED SPRITE MOVE, è necessario eseguire OLED SPRITE DRAW per aggiornare lo schermo
- È possibile usare più sprite simultaneamente, con ID diversi
- Non disegnare sprite di grandi dimensioni in quanto la memoria di ESP32 è ridotta e potrebbe crashare

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

Descrizione:

Disegna tutti gli **sprite attivi e visibili** sulla scena, in base alla loro posizione e tipo.

Esempi pratici

```
10 OLED INIT 128 32 3C  
20 OLED SPRITE NEW 0 TEXT 10 10 1 "HELLO"  
30 OLED SPRITE SHOW 0  
40 OLED SPRITE DRAW
```

Note:

- Cancella lo schermo prima di ridisegnare.
- Solo gli sprite SHOW e ACTIVE vengono mostrati.

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 32 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
```

Note:

- Per rendere effettivo lo spostamento, chiamare OLED SPRITE DRAW.

OLED SPRITE NEW

Sintassi:

OLED SPRITE NEW <id> <tipo> <x> <y> [w] [h]

Descrizione:

Crea un nuovo sprite con identificativo <id>, tipo e parametri associati.

Tipi supportati e parametri:

Tipo	Parametri	Descrizione
RECT	<x> <y> <w> <h>	Rettangolo pieno
FRAME	<x> <y> <w> <h>	Rettangolo vuoto
CIRCLE	<x> <y> <r>	Cerchio pieno
LINE	<x> <y> <dx> <dy>	Linea da (x, y) a (x+dx, y+dy)
TEXT	<x> <y> <size> "testo"	Testo statico
CHAR	<x> <y> <"char"> <size>	Singolo carattere
NUM	<x> <y> <numero> <size>	Numero intero

Esempi pratici

```
10 OLED INIT 128 32 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 FOR I = 0 TO 6
100 OLED SPRITE SHOW I
110 NEXT I
120 OLED SPRITE DRAW
130 END
```

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 SPRITE TEXT

Sintassi:

OLED SPRITE TEXT id x y size "testo"

Descrizione:

Il comando OLED SPRITE TEXT crea uno sprite testuale sul display OLED. È possibile specificare:

- id: l'identificativo numerico univoco dello sprite (da 0 a MAX_SPRITES - 1)
- x, y: coordinate di partenza del testo sullo schermo
- size: dimensione del testo (1 = normale, 2 = doppio, ecc.)
- "testo": stringa di testo da visualizzare

Lo sprite creato può essere successivamente **mostrato**, **nascosto**, **spostato**, oppure **modificato** con il comando OLED SPRITE SETTEXT.

Note:

- Il testo va messo tra virgolette doppie " "
- Gli sprite testuali sono **bufferizzati**, quindi puoi aggiornare il contenuto senza ridisegnare da zero
- Richiede un display già inizializzato con OLED INIT

Esempio pratico

```
10 OLED INIT 128 32 3C
20 OLED SPRITE TEXT 0 10 16 1 "CIAO"
30 OLED SPRITE SHOW 0
40 OLED SPRITE DRAW
50 DELAY 1000
60 OLED SPRITE SETTEXT 0 "MONDO"
70 OLED SPRITE DRAW
80 END
```

Output:

- Mostra "CIAO" al centro del display
- Dopo 1 secondo, lo sprite cambia contenuto in "MONDO" senza spostarsi

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

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 omissso).

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 (byte)** in una specifica **locazione di memoria** dell'ESP32.

È usato per accedere direttamente a **porzioni di RAM, registri hardware**, o spazi di memoria mappati.

L'indirizzo e il valore devono essere numeri interi:

- indirizzo è la posizione di memoria da modificare
- valore è un numero compreso tra 0 e 255

□ L'uso improprio di POKE può causare **blocchi** o **comportamenti anomali**, quindi è consigliato **solo a utenti esperti** o in contesti controllati.

Esempi pratici

Esempio 1 – Scrittura di un byte generico

```
10 POKE 1000, 42  
RUN
```

Output atteso:

Scrive il valore 42 all'indirizzo 1000. Nessun output a video.

Esempio 2 – Scrittura seguita da lettura con PEEK

```
10 POKE 2000, 77  
20 PRINT "VALORE IN MEMORIA: "; PEEK(2000)  
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 3000 + I, I  
30 NEXT I  
RUN
```

Output atteso:

Valori da 0 a 9 vengono scritti da 3000 a 3009.

Esempio 4 – Simulazione di buffer

```
10 FOR I = 0 TO 4
20 POKE 4000 + I, I * 10
30 NEXT I
40 FOR I = 0 TO 4
50 PRINT PEEK(4000 + I)
60 NEXT I
RUN
```

Output atteso:

```
0
10
20
30
40
```

Nota:

- Non tutti gli indirizzi sono accessibili: usare range validi documentati dal firmware
- Se si accede a zone protette o non esistenti, può verificarsi un **crash**
- Per leggere la memoria, 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

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

(oppure RENAME F\$ N\$)

Sintassi

```
RENAME "nomevecchio" "nomenuevo"  
RENAME variabile$ variabile$
```

Descrizione

Rinomina un file presente sulla **scheda SD**.
Accetta sia stringhe tra virgolette sia variabili stringa (\$).

Esempi pratici

Esempio 1 – Rinomina un file su SD

```
RENAME "log.txt" "log_old.txt"
```

Output atteso:

File renamed on SD.

Esempio 2 – Con variabili stringa

```
F$="data.csv" : N$="data_2025.csv"  
RENAME F$ N$
```

Output atteso:

File renamed on SD.

Esempio 3 – File inesistente

```
RENAME "missing.txt" "whatever.txt"
```

Output atteso:

FILE RENAME: Rename failed on SD.

Note

- Opera **solo su SD** (non su SPIFFS).
- Non crea cartelle né sposta tra directory diverse: rinomina il percorso indicato.
- I nomi sono trattati come case-sensitive a seconda del filesystem.

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

Descrizione:

Inizializza il modulo RC522, specificando i pin **SS (SDA)** e **RST**.
Gli altri pin SPI (SCK=18, MOSI=23, MISO=19 su ESP32) sono fissi.

Esempi pratici

Esempio 1 – Inizializza e segnala pronto:

```
10 RFID INIT 5 27
20 PRINT "RFID pronto"
RUN
Output atteso:
RFID pronto
```

Esempio 2 – Inizializza e controlla presenza tag:

```
10 RFID INIT 5 27
20 IF RFID PRESENT=1 THEN PRINT "C'è un tag"
RUN
```

Esempio 3 – Inizializzare con altri device SPI come display,touch ecc...

```
10 PINMODE 17 OUTPUT NOPULL      'pin cs del display ili
20 DWRITE 17 1                    'metti a HIGH il pin
30 PINMODE 4 OUTPUT NOPULL       'pin cs del touch ili
40 DWRITE 4 1                     'metti a HIGH il pin
50 RFID INIT 5 27
60 PRINT "RFID pronto"
RUN
```

Note:

- Deve essere chiamato **una sola volta** all'avvio.
- Se non lo chiami, gli altri comandi RFID falliscono.

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 DO
30 IF RFID PRESENT=1 THEN PRINT "Tag rilevato!"
40 LOOP
```

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$="CAFEBABE000000000000000000000000" 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"
```

Descrizione:

Seleziona e apre (o crea) il file JSON per il DB, caricandolo in RAM.

Esempio:

```
10 RFIDDB INIT "tags.json"  
20 PRINT "DB pronto"
```

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

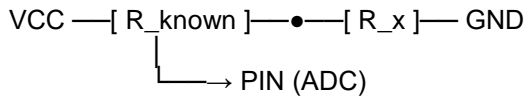
RREAD

Sintassi:

RREAD <pin> <VCC> <R_known>

Descrizione:

Calcola una **resistenza incognita** R_x misurando la tensione sul nodo del partitore nello schema:



Formula:

$$R_x = (V_{\text{node}} * R_{\text{known}}) / (VCC - V_{\text{node}})$$

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 "nomefile.bas"  
RUN "SPIFFS:nomefile.bas"  
RUN "SD:nomefile.bas"
```

Descrizione:

Il comando RUN avvia l'esecuzione del programma BASIC in memoria oppure carica e avvia un file .bas salvato nella memoria **SPIFFS** o su **scheda SD**.

- Se usato senza parametri, esegue il listato attualmente in memoria.
- Se usato con un nome di file (tra virgolette), carica il file specificato da SPIFFS o SD e lo esegue.
- Se si specifica un prefisso SPIFFS: o SD:, forza il caricamento da quella memoria.

RUN:

- Interrompe ogni programma in corso.
 - Può essere richiamato dopo un STOP o un errore.
 - Può essere incluso in un altro programma.
-

Esempi pratici

Esempio 1 – Eseguire il programma in memoria

```
10 PRINT "CIAO MONDO"  
20 END  
RUN
```

Output atteso:

CIAO MONDO

Esempio 2 – Riavvio dopo STOP

```
10 INPUT X  
20 IF X = 0 THEN STOP  
30 PRINT "VALORE: "; X
```

Se inserisci:

? 0

Poi scrivi:

RUN

Il programma riparte da capo.

Esempio 3 – Eseguire un file da SPIFFS

RUN "menu.bas"

Cerca e avvia menu.bas da SPIFFS (o SD se assente in SPIFFS).

Esempio 4 – Forzare sorgente specifica

RUN "SPIFFS:demo.bas" ' Solo da SPIFFS

RUN "SD:gioco.bas" ' Solo da SD

Note:

- Se il file specificato non esiste, viene mostrato un errore.
- Se non ci sono righe in memoria e si esegue RUN senza parametri, non succede nulla.
- I file caricati con RUN "file.bas" sovrascrivono il listato attuale.

SAVE

(o SAVE F\$)

Sintassi:

```
SAVE "nomefile"  
SAVE variabile$
```

Descrizione:

Il comando SAVE salva **il programma attualmente in memoria SD** su file

Può usare:

- un **nome file diretto** tra virgolette
- una **variabile stringa** (es: F\$) che contiene il nome del file

Il file viene salvato con estensione .bas (opzionale ma consigliata).

Esempi pratici

Esempio 1 – Salvataggio semplice

```
SAVE "demo.bas"
```

Output atteso:

Il listato BASIC in RAM viene salvato come demo.bas sulla SD o memoria interna.

Esempio 2 – Uso con variabile

```
10 LET F$ = "programma1.bas"  
20 SAVE F$  
RUN
```

Output atteso:

Il file programma1.bas viene creato con il contenuto attuale.

SAVEVAR

Sintassi:

SAVEVAR(file chiave valore)

Descrizione:

Il comando SAVEVAR salva una coppia **chiave-valore** in un file **JSON** all'interno della **scheda SD**.

È utilizzato per:

- Memorizzare impostazioni, dati utente o risultati
- Salvare valori numerici o stringhe tra diverse esecuzioni
- Archiviare dati in formato leggibile anche da PC

Il file viene creato se non esiste.

Se la chiave esiste, il valore viene sovrascritto.

☐ Il valore può essere numerico o testuale. Le stringhe vanno racchiuse tra doppi apici.

Esempi pratici

Esempio 1 – Salvare nome e livello:

```
10 SAVEVAR "config.json" "NOME", "Mario"  
20 SAVEVAR "config.json" "LIVELLO" 42
```

Output atteso: Viene creato (o aggiornato) il file config.json su SD con:

```
{  
  "NOME": "Mario",  
  "LIVELLO": 42  
}
```

Esempio 2 – Sovrascrivere un valore esistente:

```
10 SAVEVAR "config.json" "LIVELLO" 99
```

Output atteso: Il valore della chiave "LIVELLO" nel file viene aggiornato a 99.

Note:

- I valori stringa devono essere scritti tra virgolette ("testo")
- Il file è in formato JSON, compatibile con qualsiasi editor/testo
- Il file viene salvato su **SD**
- Per usare SPIFFS vedi ESAVEVAR

SCANI2C

Sintassi:

SCANI2C

Descrizione:

Esegue una scansione del bus I2C e stampa tutti gli indirizzi dei dispositivi trovati.

Esempi pratici

SCANI2C

Note:

- Utile per verificare se il display OLED è correttamente collegato.
- Il risultato appare nel monitor seriale.

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 LEDCSTATUS
```

→ 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 \geq n vengono automaticamente **detach**.

Esempi pratici

Esempio 1 – Limitare a 4 servo

```
10 SERVOMAX 4
20 LEDCSTATUS
```

→ Imposta il limite runtime a 4; “LEDCSTATUS” 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 LEDCSTATUS
```

→ L’ID 4 viene detach perché oltre il nuovo limite.

Output atteso:

Nessuna stampa se non usi LEDCSTATUS.

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.

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

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:

$\text{radianti} = \text{gradi} * (\text{PI} / 180)$

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

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 x 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:

Scrivo 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)

TAB(n)

Sintassi:

TAB(n)

Descrizione:

La funzione TAB(n) sposta il cursore alla **colonna n** della riga corrente prima di stampare il contenuto successivo.

È utile per **allineare testi** a posizioni fisse sullo schermo, come in una **tabella** o **impaginazione in colonne**.

La numerazione delle colonne parte da 1.

Se n è inferiore o uguale alla posizione attuale del cursore, il cursore va alla colonna n **sulla riga successiva**.

Esempi pratici

Esempio 1 – Allineamento semplice

```
10 PRINT "NOME"; TAB(15); "ETA"  
20 PRINT "LUCA"; TAB(15); "12"  
30 PRINT "MARCO"; TAB(15); "15"  
RUN
```

Output atteso:

NOME	ETA'
LUCA	12
MARCO	15

Esempio 2 – Uso dinamico con variabile

```
10 POSIZIONE = 20  
20 PRINT "VOCE"; TAB(POSIZIONE); "VALORE"  
RUN
```

Output atteso:

VOCE	VALORE
------	--------

Esempio 3 – Confronto con SPC(n)

```
10 PRINT "SPC:"; "A" + SPC(5) + "B"  
20 PRINT "TAB:"; "A"; TAB(10); "B"  
RUN
```

Output atteso:

```
SPC:A  B  
TAB:A  B
```

Esempio 4 – Tabulazione oltre il margine

```
10 PRINT TAB(100); "TEST"  
RUN
```

Output atteso:

La scritta "TEST" appare molto a destra o potrebbe andare a capo a seconda della larghezza del terminale.

Nota:

- TAB(n) considera la posizione **orizzontale del cursore** corrente
- Se il testo precedente supera n, la funzione avanza alla **riga successiva**
- Ideale per costruire tabelle a colonne fisse

TAN(x)

Sintassi:

TAN(x)

Descrizione:

La funzione TAN(x) restituisce la **tangente** dell'angolo x, espresso in **radianti**.
È calcolata come:

$$\text{TAN}(x) = \text{SIN}(x) / \text{COS}(x)$$

La tangente ha **valori compresi tra $-\infty$ e $+\infty$** , con **discontinuità** in corrispondenza di angoli per cui $\text{COS}(x) = 0$ (come $\pi/2$, $3\pi/2$, ecc.).

Esempi pratici

Esempio 1 – Tangente di 0 radianti

```
10 PRINT TAN(0)
RUN
```

Output atteso:

0

Esempio 2 – Tangente di 45 gradi ($\pi/4$ 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 omissso.
- 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

Descrizione:

Il comando **TIMEZONE** imposta il fuso orario (timezone) per l'orologio interno del sistema. Il valore `codice` indica il paese per scostamento in **ore** rispetto al tempo UTC. Viene applicato automaticamente anche dopo una sincronizzazione con NTP (SYNCNTP).

Esempi pratici

Esempio 1 – Impostare il fuso orario italiano (UTC+1)

```
10 TIMEZONE IT
20 SYNCNTP
RUN
```

Output atteso:

Data e ora correnti in formato locale italiano (ora solare).

Nota:

- Per visualizzare la lista dei paesi si può usare il comando LISTTIMEZONES
- Il cambiamento ha effetto immediato
- Non modifica l'orologio hardware, ma solo l'interpretazione del tempo rispetto a UTC
- Si consiglia di usare insieme a SYNCNTP per ottenere data e ora corrette

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 $\approx 30000 \mu\text{s}$ (~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\$

Descrizione:

Il comando VERIFY confronta il **programma attualmente in memoria** con il contenuto del file specificato sulla **SD**.

Serve per **verificare se il programma è stato già salvato** o è stato modificato.

Restituisce un **messaggio di corrispondenza o differenza** tra i due contenuti:

- File uguale al listato in memoria → **corrisponde**
- File diverso dal listato in memoria → **differente**
- File non trovato → **errore**

Accetta sia:

- un nome di file tra virgolette (es: "setup.bas")
- una variabile stringa con il nome file (es: F\$)

Esempi pratici

Esempio 1 – Verifica di un file identico

```
VERIFY "main.bas"
```

Output atteso:

File uguale al listato in memoria

Esempio 2 – Verifica di un file modificato

```
10 VERIFY "backup.bas"  
RUN
```

Output atteso (se diverso):

File diverso dal listato in memoria

Nota:

- Cerca il file **solo su SD**
- Non modifica nulla: è un controllo **non distruttivo**

- Utile per evitare **doppi salvataggi inutili**

VREAD

Sintassi:

```
VREAD <pin>  
VREAD <pin> <R1> <R2>
```

Descrizione:

VREAD restituisce la **tensione in Volt**:

- Con una sola forma: misura la **tensione presente al pin** (V_{pin}).
- 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 ($\approx 2 \times V_{pin}$).

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 "nome rete" "password"

Descrizione:

La funzione WIFIAP consente di creare una rete wifi da ESP.

È utile quando si vuole **creare una pagina web** visualizzabile su una rete locale per configurare impostazioni come **ora e data** dell'ESP oppure per **inviare comandi basic** come gestione GPIO.

Esempi pratici

Esempio 1 – Creazione rete wifi locale

```
10 WIFIAP "nome rete" "password"  
RUN
```

Output atteso:

Rete wifi locale creata

Esempio 2 – Creazione di una pagina web da raggiungere tramite ip locale

```
10 WIFIAP "nome rete" "password"  
20 HTML DEFAULT  
30 HTMLOBJ "<h2>Esempio di pagina web</h2>"  
60 HTMLSTART  
RUN
```

Output atteso:

Rete wifi attivata e pagina web creata su indirizzo ip locale

Esempio 3 – Creazione di una pagina web da raggiungere tramite ip locale che comanda GPIO

```
10 PINMODE 2 OUTPUT NOPULL  
20 WIFIAP "nome rete" "password"  
30 HTML DEFAULT  
40 HTMLOBJ "<h2>Controllo LED on board</h2>"  
50 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 1\")'>Accendi il LED onboard</button>"  
60 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 0\")'>Spegni il LED onboard</button>"  
70 HTMLSTART  
RUN
```

Output atteso:

Rete wifi attivata e pagina web creata su indirizzo ip locale

WIFI

Sintassi:

WIFI "ssid" "password"

Descrizione:

La funzione WIFI consente di connettere ESP ad una rete wifi.

È utile quando si vuole **creare una pagina web** visualizzabile su una rete e per configurare impostazioni come **ora e data** dell'ESP

Esempi pratici

Esempio 1 – Connessione a wifi

```
10 WIFI "nome ssid" "password"  
RUN
```

Output atteso:

Connessione a wifi attivata

Esempio 2 – Creazione di una pagina web da raggiungere tramite ip

```
10 WIFI "nome ssid" "password"  
20 HTML DEFAULT  
30 HTMLOBJ "<h2>Esempio di pagina web</h2>"  
60 HTMLSTART  
RUN
```

Output atteso:

Connessione a wifi attivata e pagina web creata su indirizzo ip

Esempio 3 – Creazione di una pagina web da raggiungere tramite ip che comanda GPIO

```
10 PINMODE 2 OUTPUT  
20 WIFI "nome rete" "password"  
30 HTML DEFAULT  
40 HTMLOBJ "<h2>Controllo LED on board</h2>"  
50 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 1\")>Accendi il LED onboard</button>"  
60 HTMLOBJ "<button onclick='fetch(\"/exec?cmd=DWRITE 2 0\")>Spegni il LED onboard</button>"  
70 HTMLSTART  
RUN
```

Output atteso:

Rete wifi attivata e pagina web creata su indirizzo ip

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 creato 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 correttamente
- 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 = WIFIDISCONNECT
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

WS BRIGHTNESS

Sintassi

WS BRIGHTNESS value

Descrizione

Imposta la **luminosità globale** (0–255) della **striscia selezionata**. Si somma con WS LBRIGHT (per-LED).

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=max. Output effettivo = raw * (LBRIGHT/255) * (BRIGHTNESS/255)

WS BUS BRIGHTNESS

Sintassi

WS BUS BRIGHTNESS value

Descrizione

Imposta la luminosità globale uguale per **tutte** le strisce nel **BUS**.

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

```
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 WS BUS DEFINE. Compatibile con WS BUS LBRIGHT

WS BUS COUNT

Sintassi

WS BUS COUNT var

Descrizione

Scriva in var la **somma dei LED** delle strisce nel **BUS**.

Esempio – Stampa conta

```
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, rifai WS BUS DEFINE

WS BUS DEFINE

Sintassi

WS BUS DEFINE id1 id2 [id3 ...]

Descrizione

Definisce l'**ordine** delle strisce concatenate nel **BUS** (indici virtuali).

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

Gli id devono esistere (WS NEW). Puoi ridefinire quando vuoi

WS BUS FILL

Sintassi

WS BUS FILL R G B

Descrizione

Imposta **tutto il BUS** a un colore (poi invia con WS BUS SHOW).

Esempio – On/Off

```
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 prima di animazioni

WS BUS LBRIGHT

Sintassi

WS BUS LBRIGHT idx value

Descrizione

Imposta la **luminosità per-LED** (0–255) del LED **virtuale** idx nel **BUS**.

Esempio – Puntatore con coda

```
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 LBRIGHT 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

```
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

WS BUS SET

Sintassi

WS BUS SET idx R G B

Descrizione

Imposta il colore del LED **virtuale** idx nel **BUS** (poi invia con WS BUS SHOW).

Esempio – Runner

```
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 sull'indice concatenato del bus

WS BUS SHOW

Sintassi

WS BUS SHOW

Descrizione

Invia i buffer di **tutte** le strisce del **BUS**.

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

Usalo in modalità BUS (meglio di WS SHOW ALL)

WS CLEAR

Sintassi

WS CLEAR

Descrizione

Spegne **subito** tutti i LED della **striscia selezionata** (buffer azzerati + invio).

Esempio – Reset

```
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, resta tutto spento finché non reimposti e invii

WS COUNT

Sintassi

WS COUNT var

Descrizione

Scrive in var il **numero di LED** della **striscia selezionata**.

Esempio – Stampa count

```
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 → errore

WS DELETE

Sintassi

WS DELETE id

Descrizione

Rimuove la striscia id e libera la memoria. Se era nel **BUS**, la rimuove e ricalcola; se era **selezionata**, azzera la selezione.

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, spegni prima con WS SELECT id + WS CLEAR. Dopo DELETE, aggiorna eventuale BUS

WS DELETE ALL

Sintassi

WS DELETE ALL

Descrizione

Spegne e rimuove tutte le strip, **svuota** il BUS, **azzerà** la selezione.

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, ricrea con WS NEW e, se serve, WS BUS DEFINE

WS FILL

Sintassi

WS FILL R G B

Descrizione

Imposta **tutti i LED** della **striscia selezionata** (poi WS SHOW).

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 singoli LED usa WS SET

WS LBRIGHT

Sintassi

WS LBRIGHT idx value

Descrizione

Imposta la **luminosità per-LED** (0–255) sulla striscia selezionata (poi WS SHOW).

Esempio – Scala

```
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 cambia il colore, solo l'intensità. Si somma con WS BRIGHTNESS

WS NEW (NeoPixel WS2812B)

Sintassi

WS NEW id pin count

Descrizione

Crea/inizializza una **striscia** con identificatore id, su pin, con count LED. Se id esiste, viene ricreata.

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)

5V stabile, GND comune, resistenza $\sim 330\Omega$ su DATA, condensatore 470–1000 μ F vicino alla strip; per strip lunghe, inietta 5V in più punti

WS SELECT

Sintassi

WS SELECT id

Descrizione

Seleziona la **striscia** su cui opereranno i comandi WS ... (non BUS).

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, i comandi falliscono

WS SET

Sintassi

WS SET idx R G B

Descrizione

Imposta il **colore** del LED idx sulla striscia selezionata (poi WS SHOW).

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

Indice 0..count-1

WS SHOW

Sintassi

WS SHOW
WS SHOW ALL

Descrizione

Invia i buffer:

- WS SHOW → **striscia selezionata**
- WS SHOW ALL → **tutte** le strisce create

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 usa WS BUS SHOW

INDICE

Introduzione a Basic32 – Interprete BASIC per ESP32	1
Installazione e Primo Avvio	3
Gestione File e Memoria.....	5
ABS(x)	6
ACS CALIB SETOFFSET	7
ACS CALIB SHOW	8
ACS CALIB ZERO	9
ACS INIT (acs712).....	10
ACS READ.....	11
ACS RMS.....	12
ACS SAMPLES.....	13
ACS SENS	14
ACS VREF	15
ACS WATCH.....	16
ADC CAL	18
AREAD(p)	20
AND, OR, NOT (Operatori Logici)	22
ASC(A\$).....	24
AUTORUN.....	26
AWRITE(p, v).....	28
BREAKPIN.....	30
BT AT	31
BT ATMODE OFF	32
BT ATMODE ON.....	33
BT AVAILABLE	34
BT END	35
BT FLUSH.....	36
BT INIT (HC-05/HC-06)	37

BT PRINT	38
BT READ	39
BT READLN.....	40
BT SEND.....	41
BT SETBAUD	42
BT SETNAME	43
BT SETPIN.....	44
BT TIMEOUT.....	45
BT VERSION.....	46
CALLFUNC	47
CHR\$(x)	48
CLS	50
CLSANSI.....	51
COPY	52
COS(x)	53
DATA	55
DATED	57
DATEM.....	58
DATEY	59
DEBUGMEM	60
DEF FN	62
DEL "file".....	64
DELN.....	65
DELVAR.....	67
DELAY n	68
DEV AUTO ON/OFF.....	70
DEV CLEAR.....	71
DEV CURSOR.....	72
DEV DISPLAY	73
DEV FONT	75

DEV ILI SET	76
DEV LCD I2C SET	77
DEV LCD PAR SET	78
DEV MAPCOLS	79
DEV MAPROWS	80
DEV OLED SET	81
DEV ON / OFF	82
DEV PS2 OFF	83
DEV PS2 ON	84
DEV PS2 SET	85
DEV SERIAL ON / OFF	86
DEV STATUS	88
DHTCALIBRESET	89
DHTCALIBSET	91
DHTCALIBSHOW	93
DHTINIT (DHT11-DHT22)	94
DHTREAD	95
DIM	96
DLEVEL(p)	98
DO	100
DO BLOCK	101
DREAD(p)	103
DWRITE(p, v)	105
DIR	107
ECOPY	108
EDEL	109
EDELVAR	110
EDIR	111
EFORMAT	112
ELISTVARS	113

ELOAD.....	114
ELOADVAR	115
ELSE	116
ERENAME.....	118
ESAVEVAR.....	119
ESAVE	120
EVERIFY	121
EXAMPLES.....	122
EXP(x).....	123
FNname(...).....	125
FOR/NEXT	127
FORMAT	129
FREEMEM.....	130
FUNC / ENDFUNC	131
...TO...STEP...NEXT	133
GET	135
GOSUB n	137
GOTO n.....	139
HTMLOBJ	141
HTMLSTART.....	142
HTTP GET.....	143
HTTP HEADERS.....	144
HTTP POST	145
IF ... THEN [ELSE]	147
ILI CIRCLE	149
ILI CLEAR	150
ILI FILLCIRCLE	151
ILI FILLRECT	152
ILI INIT (ILI9341)	153
ILI INIT TOUCH (XPT2046).....	154

ILI LED	156
ILI LINE	157
ILI PIXEL	158
ILI RECT	159
ILI SETBGCOLOR	160
ILI SPRITE CHAR	161
ILI SPRITE CLEAR	162
ILI SPRITE DELETE	163
ILI SPRITE DRAW	164
ILI SPRITE FRAME	165
ILI SPRITE HIDE	166
ILI SPRITE LINE	167
ILI SPRITE MOVE	168
ILI SPRITE NEW	169
ILI SPRITE SETCHAR	171
ILI SPRITE SETNUM	172
ILI SPRITE SETTEXT	173
ILI SPRITE SHOW	175
ILI SPRITE CLEAR	176
ILI SPRITE COLLISION	177
ILI SPRITE COLLISIONC.....	179
ILI TEXT	181
ILI TOUCH AREA.....	182
ILI TOUCH CALIBRATE	183
ILI TOUCH SPRITE.....	185
INITRTC	187
INITSD.....	188
INPUT.....	190
INT(x)	192
IP.....	194

IPAP	195
JSON GET	196
KPAVAILABLE	197
KPFLUSH.....	198
KPINIT (Matrix keypad).....	199
KPMAP	200
KPMODE.....	201
KPREAD	202
KPWAIT	203
LCD BACKLIGHT (<i>solo I²C</i>).....	204
LCD CLEAR.....	205
LCD CURSOR.....	206
LCD HOME	207
LCD I2C INIT (16X2 – 16X4).....	208
LCD PAR INIT (16X2 – 16X4).....	209
LCD PRINT	210
LCD ROW	211
LCD SCROLL	212
LCD SETCUR	213
LEDCSTATUS	214
LEFT\$(A\$, N).....	215
LEN(A\$)	217
LET.....	219
LIST.....	221
LISTVARS.....	222
LOAD	223
LOADGIT	224
LOADVAR.....	225
LISTTIMEZONES.....	226
LOG(x)	227

MEMCLEAN	229
MID\$(A\$, start, len)	231
MQTTAUTOPOLL	233
MQTTCONNECT	234
MQTTPUB	235
MQTTSUB	236
NEW	238
NOTONE	239
NOWCLR	240
NOWINIT (ESP-NOW)	242
NOWSEND	244
NRF AVAILABLE	246
NRF CONFIG	247
NRF FLUSH	248
NRF INIT (nRF24L01)	249
NRF POWERDOWN	250
NRF POWERUP	251
NRF READ	252
NRF SEND	253
NRF SET RXADDR	254
NRF SET TXADDR	255
NRF START RX	256
NRF STOP RX	257
OLED CIRCLE	258
OLEDATA / ENDOLEDATA	259
OLED CLEAR	260
OLED FILLRECT	261
OLED INIT (DISPLAY SSD1306)	262
OLED INVERT ON / OFF	263
OLED LINE	264

OLED PIXEL	265
OLED RECT	266
OLED UPDATE	267
OLED TEXT	268
OLED SPRITE COLLISION	269
OLED SPRITE COLLISIONC	271
OLED SPRITE DATA	273
OLED SPRITE DELETE	274
OLED SPRITE DRAW	275
OLED SPRITE HIDE	276
OLED SPRITE MOVE	277
OLED SPRITE NEW	278
OLED SPRITE SETCHAR	279
OLED SPRITE SETNUM	280
OLED SPRITE SETTEXT	281
OLED SPRITE SHOW	282
OLED SPRITE TEXT	283
ON x GOTO	284
PEEK	286
PINMODE	288
POKE	290
PRINT	292
READ	294
REBOOT	296
RENAME	297
RESTORE	298
RETURN	300
RFID HALT	302
RFID INIT (MFRC522)	303
RFID PRESENT	304

RFID READBLOCKABS	305
RFID READUID	306
RFID WAITUID	307
RFID WRITEBLOCKABS	308
RFIDDB ADD	309
RFIDDB CLEAR	311
RFIDDB COUNT	312
RFIDDB DELETEFILE	313
RFIDDB EXISTS	314
RFIDDB INIT	315
RFIDDB LABEL	316
RFIDDB LIST	317
RFIDDB LOAD	318
RFIDDB REMOVE	319
RFIDDB SAVE	321
RIGHT\$(A\$, N)	322
RND(x) / RND(a, b)	324
RREAD	327
RUN	329
SAVE	331
SAVEVAR	332
SCANI2C	334
SDFREE	335
SERVOATTACH	336
SERVODETACH	337
SERVOMAX	338
SERVOREAD	339
SERVOWRITE	340
SERVOWRITEMICROS	341
SETDATE	342

SETTIME	343
SIN(x)	344
SPC(n)	346
SPIFREE	348
SR595 CLEAR	349
SR595 FILL	350
SR595 INIT (SHIFT REGISTER 74HC595)	351
SR595 SET	353
SR595 SHOW	354
SR595 WRITE	355
STARTFUNC / STOPFUNC	356
STOP, CONT, END	358
STR\$(x) o STR\$(x, n)	360
SYNCTP	362
TAB(n)	363
TAN(x)	365
TCSFILTER	367
TCSINIT (Sensore colore TCS230/TCS3200)	368
TCSLED	369
TCSREAD	370
TCSRGB	372
TCSSCALE	373
TI	374
TI\$	376
TIMEH	378
TIMEM	379
TIMES	380
TIMEZONE	381
TONE	382
ULTRA INIT (SENSORE ULTRASUONI HC-SR04)	384

ULTRA READ	385
ULTRA TIMEOUT	386
VAL(A\$)	387
VERIFY	389
VREAD.....	391
WAIT n	392
WIFIAP	394
WIFI.....	396
WIFICONNECTED	398
WIFIAPCONNECTED	399
WIFIDISCONNECT	400
WIFIAPDISCONNECT	401
WS BRIGHTNESS	402
WS BUS BRIGHTNESS.....	403
WS BUS COUNT.....	404
WS BUS DEFINE	405
WS BUS FILL.....	406
WS BUS LBRIGHT	407
WS BUS SET	409
WS BUS SHOW	410
WS CLEAR	411
WS COUNT	412
WS DELETE.....	413
WS DELETE ALL.....	414
WS FILL	415
WS LBRIGHT	416
WS NEW (NeoPixel WS2812B).....	417
WS SELECT.....	418
WS SET.....	419
WS SHOW.....	420

INDICE 421