Versione 1.0

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

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 (multipiattaforma)

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"
```

Comandi BASIC supportati

Basic32 implementa una versione compatta ma potente del linguaggio BASIC, pensata per l'interazione diretta con l'hardware ESP32. Qui sotto trovi i comandi organizzati per tipologia, con esempi pratici.

Gestione Programma

Comando	Descrizione	Esempio
RUN	Avvia il programma memorizzato	RUN
LIST	Mostra tutte le righe del programma in memoria	LIST
NEW	Cancella tutte le righe del programma in memoria	NEW

Input/Output

Comando	Descrizione	Esempio
PRINT	Stampa un valore o messaggio sulla seriale	PRINT "CIAO"
INPUT	Chiede un valore all'utente	INPUT A
GET	Legge un singolo carattere (non attende Enter)	A = GET

Variabili e Espressioni

Comando	Descrizione	Esempio
LET	Assegna un valore a una variabile	LET A = 10
DIM	Definisce un array	DIM A(10)
DEF FN	Definisce una funzione personalizzata	DEF FNADD(X,Y)=X+Y
FNname(...)	Chiama una funzione definita	PRINT FNADD(2,3)

Controllo di Flusso

Comando	Descrizione	Esempio
IF ... THEN	Esegue condizionalmente una linea	IF A=10 THEN PRINT "OK"
ELSE	Aggiunge alternative a IF	IF A=0 THEN ... ELSE PRINT "NO"
GOTO n	Salta alla riga n del programma	GOTO 100
GOSUB n	Chiama una subroutine	GOSUB 200
RETURN	Torna da una subroutine	RETURN
ON x GOTO...	Salto condizionale su valore	ON A GOTO 100,200,300
FOR/NEXT	Inizia e termina un ciclo	FOR I=1 TO 10 ... NEXT I
STEP	Specifica l'incremento nel ciclo FOR	FOR I=1 TO 10 STEP 2

Controllo GPIO e Hardware

Comando	Descrizione	Esempio
PINMODE(p,m,r)	Imposta il pin p come INPUT/OUTPUT con pull-up/none	PINMODE 2, OUTPUT, NOPULL
DWRITE(p,v)	Scrive HIGH o LOW su pin	DIGITALWRITE 2, HIGH
DREAD(p)	Legge lo stato digitale di un pin	A = DIGITALREAD(4)
AREAD(p)	Legge valore analogico (0–4095)	A = ANALOGREAD(36)

Gestione Dati e Costanti

Comando	Descrizione	Esempio
DATA	Elenco di dati da leggere	DATA 10, 20, 30
READ	Legge il prossimo valore da DATA	READ A
RESTORE	Riporta il puntatore dei dati all'inizio	RESTORE

Funzioni Matematiche

Funzione	Descrizione	Esempio
ABS(x)	Valore assoluto	PRINT ABS(-3)
INT(x)	Parte intera	PRINT INT(3.9)
SIN(x)	Seno di x (in gradi)	PRINT SIN(30)
COS(x)	Coseno di x	PRINT COS(60)
TAN(x)	Tangente di x	PRINT TAN(45)
LOG(x)	Logaritmo naturale	PRINT LOG(10)
EXP(x)	e elevato a x	PRINT EXP(1)
RND(a) - RND(a,b)	Numero casuale	PRINT RND(10) o RND(1,6)

Comandi File System (SPIFFS/SD)

Comando	Descrizione	Esempio
SAVE "file"	Salva programma su memoria SD	SAVE "prog1.bas"
LOAD "file"	Carica programma da memoria SD	LOAD "prog1.bas"
DIR	Elenca i file presenti nella memoria SD	DIR
DEL “file”	Cancella un file dalla memoria SD	DEL “vecchio.bas”
VERIFY “file”	Verifica se il codice in memoria è uguale a quello nel file nella memoria SD	VERIFY “prog1.bas”
ESAVE "file"	Salva programma su memoria SPIFFS	ESAVE "prog1.bas"
ELOAD "file"	Carica programma da memoria SPIFFS	ELOAD "prog1.bas"
EDIR	Elenca i file presenti nella memoria SPIFFS	EDIR

Comando	Descrizione	Esempio
EDEL “file”	Cancella un file dalla memoria SPIFFS	EDEL “vecchio.bas”
EVERIFY “file”	Verifica se il codice in memoria è uguale a quello nel file nella memoria SPIFFS	EVERIFY “prog1.bas”

Comandi Aggiuntivi (tempo, aiuto, ecc.)

Comando	Descrizione	Esempio
CLS	Pulisce lo schermo (seriale)	CLS
CLSANSI	Pulisce lo schermo (seriale) tramite comando ANSI	CLSANSI
HELP	Elenca comandi disponibili	HELP

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

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 BASSA" ELSE PRINT "LUCE OK"
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: B = 2
20 IF A = 1 AND B = 2 THEN PRINT "ENTRAMBI VERI"
RUN
```

Output atteso:

```
ENTRAMBI VERI
```

Esempio 2 – Uso con OR

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

Output atteso:

```
ALMENO UNO È DIVERSO DA ZERO
```

Esempio 3 – Uso di NOT

```
10 A = 0
```

```
20 IF NOT A THEN PRINT "A È ZERO"
RUN
```

Output atteso:

```
A È ZERO
```

Esempio 4 – Maschera di bit con AND

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

Output atteso:

```
RISULTATO: 4
```

Esempio 5 – Uso in assegnazione logica

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

Output atteso:

```
1
```

Nota:

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

ASC(A$)

Sintassi:

ASC(stringa$)

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

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

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

Esempi pratici

Esempio 1 – Codice ASCII di una lettera

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

Output atteso:

```
65
```

Esempio 2 – Confronto con una lettera specifica

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

Output atteso:

```
È Z
```

Esempio 3 – Da carattere a codice e ritorno

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

Output atteso:

C

Esempio 4 – 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

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.

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

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)

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)

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
```

DREAD(p)

Sintassi:

DREAD(pin)

Descrizione:
La funzione DREAD(p) legge il **valore logico** (digitale) presente sul **pin p** dell'ESP32.
Restituisce:

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

È utile per leggere il **livello logico di pulsanti, interruttori o sensori digitali**.
Assicurati che il pin sia stato correttamente configurato in modalità **INPUT** tramite PINMODE.

Esempi pratici

Esempio 1 – Lettura diretta da un pin

```
10 PINMODE 4, INPUT, NOPULL
20 V = DREAD(4)
30 PRINT "STATO DEL PIN 4: "; V
RUN
```

Output atteso:

```
STATO DEL PIN 4: 0     (oppure 1, a seconda del collegamento)
```

Esempio 2 – Verifica pressione di un pulsante
→ Supponendo un pulsante collegato tra **pin 5** e **GND**, con **PULLUP attivo**:

```
10 PINMODE 5, INPUT, PULLUP
20 IF DREAD(5) = 0 THEN PRINT "PULSANTE PREMUTO" ELSE PRINT "PULSANTE RILASCIATO"
RUN
```

Output atteso (quando il pulsante è premuto):

```
PULSANTE PREMUTO
```

Esempio 3 – Lettura continua in loop

```
10 PINMODE 2, INPUT, NOPULL
20 DO
30 PRINT "PIN 2 = "; DREAD(2)
40 WAIT 500
50 LOOP
```

RUN

Output atteso:

PIN 2 = 0
PIN 2 = 1
...

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
30 DWRITE 2, 1
40 WAIT 500
50 DWRITE 2, 0
60 WAIT 500
70 LOOP
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

EDEL “file”

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)

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.

ELOAD "file"

(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.

ESAVE "file"

(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.

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 INPUT V
20 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

EVERIFY "file"

(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 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**

EXP(x)

Sintassi:

EXP(x)

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

Il parametro x può essere positivo, negativo o zero.

Esempi pratici

Esempio 1 – Calcolare e^1

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

Output atteso:

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

Esempio 2 – e elevato alla seconda

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

Output atteso:

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

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

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

Output atteso:

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

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

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

Output atteso:

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

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
```

...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 PRINT "PREMI TASTI (Q PER USCIRE):"
20 DO
30 C = GET
40 IF C <> -1 THEN PRINT "TASTO: "; C
50 IF C = 81 THEN END
60 LOOP
RUN
```

Output atteso:

```
PREMI TASTI (Q PER USCIRE):
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
40 C = GET
50 IF C = 49 THEN DWRITE 2, 1
60 IF C = 48 THEN DWRITE 2, 0
70 LOOP
RUN
```

Output atteso:

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

Nota:

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

GOSUB n

Sintassi:

GOSUB numero_riga

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

È utile per:

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

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

Esempi pratici

Esempio 1 – Subroutine che stampa una riga

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

Output atteso:

```
SUBROUTINE ESEGUITA
PROGRAMMA PRINCIPALE
```

Esempio 2 – Chiamare la stessa subroutine più volte

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

Output atteso:

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

ESECUZIONE NUMERO: 3

Esempio 3 – Subroutine per calcolo riutilizzabile

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

Output atteso (es. input 3, 4):

RISULTATO: 12

Esempio 4 – Errore comune da evitare

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

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

Corretto invece con:

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

GOTO n

Sintassi:

GOTO numero_riga

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

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

Esempi pratici

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

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

Output atteso:

```
INIZIO
DOPO IL SALTO
```

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

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

Output atteso:

```
1
2
3
4
5
```

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

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

Output atteso (se premi 1):

```
INIZIO GIOCO
```

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

```
10 PRINT "PREMI CTRL+C PER USCIRE"
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.

HELP

Sintassi:

HELP

Descrizione:
Il comando HELP mostra a schermo l'elenco dei **comandi BASIC32 supportati** direttamente sul terminale seriale.
È utile per ricordare rapidamente la sintassi dei comandi, le funzionalità disponibili e le parole chiave implementate nel sistema.

Visualizza:

- comandi principali (PRINT, INPUT, GOTO, ecc.)
- comandi di controllo I/O (PINMODE, DIGITALWRITE, ANALOGREAD, ecc.)
- struttura di controllo (IF, FOR, GOSUB, ecc.)
- comandi di gestione file e memoria (SAVE, LOAD, FILES, ecc.)
- comandi di sistema (RUN, LIST, NEW, END, ecc.)

Non richiede parametri.

Esempi pratici

Esempio – Visualizzare l'elenco comandi disponibili

HELP

Output atteso (parziale):

Available BASIC Commands:

PRINT expr
INPUT var
GET
LET var = expr
IF cond THEN ...
GOTO line
GOSUB line
RETURN
FOR ... TO ... STEP ...
NEXT
...

Nota:

- Il comando HELP stampa direttamente sul terminale: **non restituisce valori**.
- Utile da inserire all'inizio dei programmi interattivi o nei file di esempio.

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

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
- Non supporta input su stessa riga come INPUT "TESTO"; A (per ora)

INT(x)

Sintassi:

INT(x)

Descrizione:
La funzione INT(x) restituisce la **parte intera** di un numero x, **tronca i decimali** e **arrotonda verso lo zero**.
È utile per convertire un numero decimale in intero in modo controllato, ad esempio per gestire cicli, indici di array, arrotondamenti personalizzati.

- INT(4.7) ⟶ 4
- INT(-2.3) ⟶ -2

 Non confondere con un arrotondamento: INT **non** arrotonda per eccesso o difetto — taglia semplicemente i decimali.

Esempi pratici

Esempio 1 – Parte intera di un numero positivo

```
10 A = 5.89
20 PRINT "INT(5.89) = "; INT(A)
RUN
```

Output atteso:

```
INT(5.89) = 5
```

Esempio 2 – Parte intera di un numero negativo

```
10 PRINT "INT(-3.99) = "; INT(-3.99)
RUN
```

Output atteso:

```
INT(-3.99) = -3
```

Esempio 3 – Uso per accedere a indici di array
⟶ Elimina il decimale da una divisione e usa il risultato come indice:

```
10 DIM A(10)
20 X = 5.7
30 A(INT(X)) = 99
40 PRINT A(5)
RUN
```

Output atteso:

99

Esempio 4 – Uso in un ciclo per numeri casuali interi
→ Genera 10 numeri casuali da 0 a 9:

```
10 FOR I = 1 TO 10
20 PRINT INT(RND(1) * 10)
30 NEXT I
RUN
```

Output atteso:
(esempio)

```
7
2
9
0
5
...
```

LEFT$(A$, N)

Sintassi:

LEFT$(stringa$, N)

Descrizione:

La funzione LEFT$ restituisce una **sottostringa** contenente i **primi N caratteri** della stringa A$.
Se N è maggiore della lunghezza della stringa, viene restituita **l'intera stringa**.

Utile per:

- Analisi o taglio di stringhe
- Parsing di input
- Verifica di prefissi o codici

Esempi pratici

Esempio 1 – Primi 4 caratteri

```
10 A$ = "BASIC32"
20 PRINT LEFT$(A$, 4)
RUN
```

Output atteso:

```
BASI
```

Esempio 2 – Uso in IF per riconoscere un comando

```
10 COM$ = "LOAD file.bas"
20 IF LEFT$(COM$, 4) = "LOAD" THEN PRINT "COMANDO DI CARICAMENTO"
RUN
```

Output atteso:

```
COMANDO DI CARICAMENTO
```

Esempio 3 – Valore di N maggiore della lunghezza

```
10 T$ = "ESP"
20 PRINT LEFT$(T$, 10)
RUN
```

Output atteso:

ESP

Esempio 4 – Sottostringa con N = 0

```
10 A$ = "TEST"
20 PRINT LEFT$(A$, 0)
RUN
```

Output atteso:

(empty string)

Esempio 5 – Lettura di codice numerico da inizio riga

```
10 RIGA$ = "12345: PRINT 'CIAO'"
20 CODICE$ = LEFT$(RIGA$, 5)
30 PRINT "CODICE = "; CODICE$
RUN
```

Output atteso:

CODICE = 12345

Nota:

- N deve essere ≥ 0
- Funziona solo con variabili stringa ($)
- Combinabile con RIGHT$, MID$, LEN, ASC, CHR$, ecc.

LEN(A$)

Sintassi:

LEN(stringa$)

Descrizione:
La funzione LEN restituisce la **lunghezza** (in caratteri) di una **stringa**.
Conta **ogni carattere**, inclusi spazi, simboli, numeri, lettere, ecc.

È utile per:

- Verificare input dell'utente
- Controllare se una stringa è vuota
- Lavorare con substringhe

Esempi pratici

Esempio 1 – Lunghezza di una stringa

```
10 A$ = "CIAO"
20 PRINT "LUNGHEZZA: "; LEN(A$)
RUN
```

Output atteso:

```
LUNGHEZZA: 4
```

Esempio 2 – Stringa con spazi

```
10 T$ = "CIAO MONDO"
20 PRINT LEN(T$)
RUN
```

Output atteso:

```
10
```

Esempio 3 – Stringa vuota

```
10 V$ = ""
20 PRINT LEN(V$)
RUN
```

Output atteso:

```
0
```

Esempio 4 – Uso in condizione IF

```
10 INPUT "INSERISCI TESTO: "; T$
20 IF LEN(T$) = 0 THEN PRINT "NESSUN DATO INSERITO"
RUN
```

Output atteso (se si preme solo INVIO):

```
NESSUN DATO INSERITO
```

Esempio 5 – Contatore caratteri

```
10 MSG$ = "BASIC32"
20 PRINT "NUMERO DI CARATTERI: "; LEN(MSG$)
RUN
```

Output atteso:

```
NUMERO DI CARATTERI: 7
```

Nota:

- LEN funziona **solo con variabili stringa** ($)
- Non restituisce errori se la stringa è vuota: ritorna 0
- Combinabile con LEFT$, RIGHT$, MID$, CHR$, ASC

LET

Sintassi:

```
LET variabile = espressione
```

oppure semplicemente:

```
variabile = espressione
```

Descrizione:
Il comando LET serve per **assegnare un valore a una variabile**, sia numerica che stringa ($).
È **opzionale**: puoi ometterlo e scrivere direttamente l'assegnazione (come nei BASIC più moderni).

Può assegnare:

- costanti numeriche
- stringhe
- risultati di espressioni o funzioni
- espressioni condizionali

Esempi pratici

Esempio 1 – Assegnazione numerica semplice

```
10 LET A = 5
20 PRINT A
RUN
```

Output atteso:

```
5
```

Esempio 2 – Uso senza LET (forma abbreviata)

```
10 B = 10 * 2
20 PRINT B
RUN
```

Output atteso:

```
20
```

Esempio 3 – Assegnazione stringa

```
10 LET NOME$ = "LUCA"
```

```
20 PRINT "CIAO "; NOME$
RUN
```

Output atteso:

```
CIAO LUCA
```

Esempio 4 – Con funzioni
→ Assegnare a una variabile il risultato di una funzione:

```
10 X = SQR(49)
20 PRINT "RADICE: "; X
RUN
```

Output atteso:

```
RADICE: 7
```

Esempio 5 – Assegnazione condizionale
→ Usa IF per assegnare valori diversi:

```
10 A = 3
20 IF A < 5 THEN LET RISULTATO = 1 ELSE LET RISULTATO = 0
30 PRINT RISULTATO
RUN
```

Output atteso:

```
1
```

Nota:

- Non è possibile assegnare array direttamente (A(1) = ...) se non prima di un DIM
- Puoi usare LET per migliorare la leggibilità, anche se non è obbligatorio

LIST

Sintassi:

LIST

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

LOAD "file"

(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).

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.

- LOG(1) = 0
- LOG(e) = 1
- LOG(x) è **definito solo per x > 0**

 Se x è zero o negativo, il risultato non è valido e può generare errore o valore indefinito.

Esempi pratici

Esempio 1 – Logaritmo naturale di 1

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

Output atteso:

```
LOG(1) = 0
```

Esempio 2 – Logaritmo di e (circa 2.71828)

```
10 PRINT "LOG(E) = "; LOG(2.71828)
RUN
```

Output atteso:

```
LOG(E) = 1
```

Esempio 3 – Logaritmo di un valore maggiore

```
10 X = 10
20 PRINT "LOG(10) = "; LOG(X)
RUN
```

Output atteso:

```
LOG(10) = 2.30258
```

Esempio 4 – Uso in formula combinata
→ Calcolo della funzione f(x) = LOG(x) * x

```
10 INPUT A
20 PRINT "f(A) = "; LOG(A) * A
RUN
```

Output atteso (es. input 5):

```
f(A) = 8.047
```

Nota:

- Per calcolare logaritmi in base 10, puoi usare:

  LOG10(X) = LOG(X) / LOG(10)

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.

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.

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(p, m, r)

Sintassi:

PINMODE(pin, modo, resistenza)

Descrizione:

PINMODE configura il comportamento di un **pin GPIO** dell'ESP32, specificando se deve funzionare in **ingresso** (INPUT) o **uscita** (OUTPUT), e se deve usare una **resistenza di pull-up/down**.

- pin: numero del GPIO (es. 2, 4, 5, ecc.)
- modo: INPUT oppure OUTPUT
- resistenza: NOPULL, PULLUP, PULLDOWN

Esempi pratici

Esempio 1 – Impostare un pin come OUTPUT senza resistenze

→ Per accendere un LED collegato al pin 2:

```
10 PINMODE 2, OUTPUT, NOPULL
20 DWRITE 2, 1
RUN
```

Output atteso:

Il pin GPIO2 viene configurato come uscita e impostato su livello alto.

Esempio 2 – Impostare un pin come INPUT con pull-up

→ Per leggere un pulsante collegato tra pin e GND:

```
10 PINMODE 5, INPUT, PULLUP
20 IF DREAD(5) = 0 THEN PRINT "PREMUTO" ELSE PRINT "RILASCIATO"
RUN
```

Output atteso:

Stampa lo stato del pulsante in base alla lettura.

Esempio 3 – Impostare un pin in input senza resistenze interne

→ Per leggere segnali da sensori che hanno già pull-up/pull-down esterni:

```
10 PINMODE 4, INPUT, NOPULL
20 PRINT DREAD(4)
RUN
```

Esempio 4 – Combinare PINMODE con un ciclo
→ Legge in continuazione il valore di un pin configurato in input:

```
10 PINMODE 13, INPUT, PULLDOWN
20 DO
30 PRINT DREAD(13)
40 WAIT 500
50 LOOP
RUN
```

Output atteso:
Stampa ripetuta del livello logico del pin 13 ogni 500 ms.

Nota:

- Sempre necessario chiamare PINMODE prima di DIGITALREAD o DIGITALWRITE
- Le resistenze interne (PULLUP e PULLDOWN) sono utili quando non presenti esternamente

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

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

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

RUN

Sintassi:

```
RUN
```

Descrizione:
Il comando RUN avvia **l'esecuzione del programma BASIC caricato in memoria** partendo dalla **prima riga disponibile** (quella col numero di riga più basso).
Interrompe qualsiasi programma in corso e **riparte da capo**.

- Utilizzabile anche dopo un comando STOP o errore
- Non richiede parametri
- Può essere digitato manualmente da terminale oppure inserito in un altro programma

---

Esempi pratici

Esempio 1 – Eseguire un programma

```
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
RUN
```

Output atteso (se X = 0):

```
? 0
(STOP attivo)
```

Poi:

```
RUN
```

Output:
Programma riparte da capo chiedendo di nuovo l'input.

Esempio 3 – Dopo modifica di righe

```
10 PRINT "PRIMA VERSIONE"
RUN
```

Poi:

```
10 PRINT "VERSIONE AGGIORNATA"
RUN
```

Output atteso:

```
VERSIONE AGGIORNATA
```

Nota:

- Se non ci sono righe in memoria, `RUN` non fa nulla

SAVE "file"

(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.

SETDATE 2025,6,15

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 14,30,0

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:

radianti = gradi * (PI / 180)

Esempi pratici

Esempio 1 – Seno di 0 radianti

```
10 PRINT SIN(0)
RUN
```

Output atteso:

```
0
```

Esempio 2 – Seno di PI/2 (~1.5708 radianti)

```
10 PRINT SIN(3.14159 / 2)
RUN
```

Output atteso:

```
1
```

Esempio 3 – Calcolo del seno da gradi
→ Angolo di 30°

```
10 G = 30
20 R = G * 3.14159 / 180
30 PRINT "SIN(30°) = "; SIN(R)
RUN
```

Output atteso:

```
SIN(30°) = 0.5
```

Esempio 4 – Tabella dei seni per angoli da 0 a 90°

```
10 FOR A = 0 TO 90 STEP 15
20 R = A * 3.14159 / 180
30 PRINT "SIN("; A; "°) = "; SIN(R)
40 NEXT A
RUN
```

Output atteso:

```
SIN(0°) = 0
SIN(15°) = 0.2588
SIN(30°) = 0.5
SIN(45°) = 0.7071
SIN(60°) = 0.8660
SIN(75°) = 0.9659
SIN(90°) = 1
```

Nota:

- Il risultato è compreso tra -1 e +1
- Per altre funzioni trigonometriche usa COS(x), TAN(x), ATN(x)

SPC(n)

Sintassi:

SPC(n)

Descrizione:
La funzione SPC(n) genera una **stringa di n spazi**, utile per **formattare l'output**, **allineare testi**, o **creare margini visivi** nel terminale.

Può essere utilizzata:

- All'interno di PRINT per creare spaziature
- Per costruire righe con colonne ben separate
- In combinazione con altre stringhe

Esempi pratici

Esempio 1 – Spazio tra due parole

```
10 PRINT "CIAO" + SPC(5) + "MONDO"
RUN
```

Output atteso:

```
CIAO     MONDO
```

Esempio 2 – Allineamento su più righe

```
10 PRINT "NOME" + SPC(10) + "ETA'"
20 PRINT "LUCA" + SPC(11) + "12"
30 PRINT "MARCO" + SPC(9) + "15"
RUN
```

Output atteso:

```
NOME         ETA'
LUCA         12
MARCO         15
```

Esempio 3 – Uso dinamico

```
10 FOR I = 1 TO 5
20 PRINT SPC(I) + "TEST"
30 NEXT I
RUN
```

Output atteso:

```
TEST
  TEST
   TEST
    TEST
     TEST
```

Esempio 4 – Rientro fisso

```
10 INDENT = 8
20 PRINT SPC(INDENT) + "RIGA FORMATTA"
RUN
```

Output atteso:

```
        RIGA FORMATTA
```

Nota:

- Se `n = 0`, non viene generato alcuno spazio
- Se `n` è maggiore della larghezza del terminale, il testo potrebbe andare a capo
- Combinabile con `TAB(n)` per posizionamenti più precisi

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:

TAN(x) = SIN(x) / COS(x)

La tangente ha **valori compresi tra -∞ e +∞**, con **discontinuità** in corrispondenza di angoli per cui COS(x) = 0 (come π/2, 3π/2, ecc.).

Esempi pratici

Esempio 1 – Tangente di 0 radianti

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

Output atteso:

```
0
```

Esempio 2 – Tangente di 45 gradi (π/4 radianti)

```
10 PI = 3.14159
20 PRINT TAN(PI / 4)
RUN
```

Output atteso:

```
1
```

Esempio 3 – Tangente di 60 gradi
→ Calcolo da gradi a radianti

```
10 A = 60
20 R = A * 3.14159 / 180
30 PRINT "TAN("; A; "°) = "; TAN(R)
RUN
```

Output atteso:

```
TAN(60°) = 1.732
```

Esempio 4 – Valori critici (attenzione alla discontinuità)
→ A 90° la tangente tende all'infinito

```
10 PI = 3.14159
20 PRINT TAN(PI / 2)
RUN
```

Output atteso:

(grande valore o errore numerico)

Nota:

- L’argomento x deve essere in **radianti**
- Per evitare errori, evitare angoli in cui COS(x) = 0
- Usare con SIN(x) e COS(x) per rappresentazioni geometriche

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!
```

Esempio 4 – Uso per lampeggio a intervalli

```
10 T = TI
20 PRINT "☼"
30 WHILE TI < T + 100: WEND
40 PRINT " "
50 WHILE TI < T + 200: WEND
60 GOTO 10
RUN
```

Output atteso:
Simulazione di lampeggio con "☼" ogni 0.1 secondi.

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 codice

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

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 "file"

(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**

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
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

Programmi didattici ed esempi completi

Benvenuto nella sezione **più pratica e divertente** di questa guida: qui metteremo finalmente le mani sul **linguaggio BASIC32** per scrivere **programmi reali**, ricchi di funzionalità e significato.

In questo capitolo, troverai una raccolta di **esempi completi**, organizzati per difficoltà e **mirati a testare uno o più comandi alla volta**, con l'obiettivo di aiutarti a:

- Comprendere il funzionamento reale di ogni istruzione
- Allenarti scrivendo codice che funziona davvero
- Divertirti con mini giochi, utility e programmi interattivi
- Imparare a lavorare con memoria, file, input/output e I/O digitale

---

Come usare questa sezione

Ogni programma è strutturato in modo da:

- Mostrare il **codice completo** pronto per essere copiato e incollato nel terminale
- Fornire una **spiegazione dettagliata** delle istruzioni usate
- Presentare l'**output atteso**
- Proporre **modifiche facili** per personalizzarlo o estenderlo

---

A chi è rivolta?

Questa sezione è pensata sia per chi:

- Sta iniziando a programmare da zero
- Vuole imparare il funzionamento interno dell'ESP32
- Ama imparare facendo, modificando e riprovando
- Vuole una guida tipo libro di testo scolastico ma... **più viva e giocosa!**

Cosa ti serve?

Solo:

- La tua **scheda ESP32** con il firmware BASIC32 caricato
- Un **terminale seriale** per scrivere ed eseguire i programmi
- Curiosità, fantasia e voglia di imparare

Preparati a partire!
Nelle prossime pagine troverai programmi che coprono tutto: **dalla stampa a video** fino alla **gestione file**, **logica condizionale**, **funzioni**, **timer**, **input**, e persino il **controllo dei pin digitali** per far lampeggiare un LED.

Livello 1 – Basi della Programmazione

ESEMPIO 1 — Scrivi il tuo nome

Comandi usati: PRINT, INPUT, LET

```
10 INPUT "COME TI CHIAMI? "; N$
20 PRINT "CIAO "; N$; "!"
```

Spiegazione:

- INPUT chiede il nome all'utente e lo salva in N$
- PRINT lo saluta usando la variabile

Output atteso:

```
COME TI CHIAMI? ? Mario
CIAO Mario!
```

Prova a modificare: aggiungi un secondo input per chiedere anche il cognome.

ESEMPIO 2 — Somma due numeri

Comandi usati: INPUT, LET, PRINT

```
10 INPUT "INSERISCI IL PRIMO NUMERO: "; A
20 INPUT "INSERISCI IL SECONDO NUMERO: "; B
30 LET C = A + B
40 PRINT "LA SOMMA È: "; C
```

Spiegazione:

- Acquisisce due numeri e li somma
- LET è opzionale, ma qui chiarisce l'assegnazione

Prova a modificare: cambia il + in * per fare una moltiplicazione.

ESEMPIO 3 — Conto da 1 a 10

Comandi usati: FOR, NEXT, PRINT

```
10 FOR I = 1 TO 10
20 PRINT I
30 NEXT I
```

Spiegazione:

- Il ciclo FOR ripete il codice da 1 a 10
- Stampa ogni numero su una riga

Prova a modificare: prova con STEP 2 per stampare solo i numeri dispari.

ESEMPIO 4 — Conta alla rovescia con ritardo

Comandi usati: FOR, DELAY, PRINT

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

Spiegazione:

- Conta da 5 a 1
- DELAY 1000 mette una pausa di 1 secondo tra i numeri

Prova a modificare: metti DELAY 500 per una conta più veloce.

ESEMPIO 5 — Orario dopo 3 secondi

Comandi usati: TI$, DELAY, PRINT

```
10 PRINT "ORARIO INIZIALE: "; TI$
20 DELAY 3000
30 PRINT "DOPO 3 SECONDI: "; TI$
```

Spiegazione:

- TI$ mostra il tempo in formato hhmmss
- Dopo un ritardo, viene mostrato il nuovo orario

Prova a modificare: usa TI per vedere il tempo in centesimi di secondo.

Livello 2 – Logica, Condizioni e Interazione

ESEMPIO 6 — Indovina il numero

Comandi usati: RND, INT, INPUT, IF...THEN, GOTO, PRINT

```
10 PRINT "INDOVINA IL NUMERO DA 1 A 100"
20 TARGET = INT(RND(100)) + 1
30 INPUT "IL TUO TENTATIVO: "; N
40 IF N = TARGET THEN PRINT "COMPLIMENTI!" : GOTO 100
50 IF N < TARGET THEN PRINT "TROPPO BASSO"
60 IF N > TARGET THEN PRINT "TROPPO ALTO"
70 GOTO 30
100 PRINT "FINE GIOCO"
```

Spiegazione:

- Genera un numero casuale da 1 a 100
- L'utente tenta finché non lo indovina

Esercizio: Aggiungi un contatore di tentativi.

ESEMPIO 7 — Calcolatrice base

Comandi usati: INPUT, VAL, PRINT, IF...THEN, STR$

```
10 INPUT "NUMERO 1: "; A$
20 INPUT "OPERATORE (+, -, *, /): "; OP$
30 INPUT "NUMERO 2: "; B$
40 A = VAL(A$)
50 B = VAL(B$)
60 IF OP$ = "+" THEN PRINT "RISULTATO: "; A + B
70 IF OP$ = "-" THEN PRINT "RISULTATO: "; A - B
80 IF OP$ = "*" THEN PRINT "RISULTATO: "; A * B
90 IF OP$ = "/" THEN IF B <> 0 THEN PRINT "RISULTATO: "; A / B ELSE PRINT "DIVISIONE PER 0!"
```

Spiegazione:

- Converte input stringa in numeri
- Esegue l’operazione desiderata

Esercizio: Aggiungi supporto a ^ o MOD.

ESEMPIO 8 — Menu testuale

Comandi usati: PRINT, INPUT, IF, GOTO, CLS

```
10 CLS
20 PRINT "MENU:"
30 PRINT "1. SALUTO"
40 PRINT "2. ORARIO"
50 PRINT "3. USCITA"
60 INPUT "SCEGLI: "; S
70 IF S = 1 THEN PRINT "CIAO!" : GOTO 20
80 IF S = 2 THEN PRINT "SONO LE: "; TI$ : GOTO 20
90 IF S = 3 THEN END
100 GOTO 20
```

Spiegazione:

- Menu che torna sempre alla scelta iniziale
- Ogni opzione fa qualcosa di diverso

Esercizio: Aggiungi opzione per sommare due numeri.

ESEMPIO 9 — Conversione gradi/radianti

Comandi usati: INPUT, PRINT, PI, SIN, COS, TAN

```
10 INPUT "INSERISCI ANGOLO IN GRADI: "; G
20 R = G * 3.1416 / 180
30 PRINT "SENO: "; SIN(R)
40 PRINT "COSENO: "; COS(R)
50 PRINT "TANGENTE: "; TAN(R)
```

Spiegazione:

- Converte l’angolo in radianti
- Usa le funzioni trigonometriche

Esercizio: Aggiungi ATN per calcolare l’arcotangente.

ESEMPIO 10 — Verifica password

Comandi usati: INPUT, IF, LEFT$, LEN

```
10 PRINT "INSERISCI LA PASSWORD:"
20 INPUT ""; P$
30 IF P$ = "ABC123" THEN PRINT "ACCESSO CONCESSO" : END
40 IF LEN(P$) < 6 THEN PRINT "TROPPO CORTA"
50 IF LEFT$(P$, 3) <> "ABC" THEN PRINT "DEVE INIZIARE CON 'ABC'"
60 GOTO 10
```

Spiegazione:

- Verifica lunghezza e prefisso della password

Esercizio: Modifica per permettere massimo 3 tentativi.

ESEMPIO 11 — Comparatore di stringhe ASCII

Comandi usati: INPUT, ASC, CHR$, IF

```
10 INPUT "INSERISCI DUE LETTERE: "; A$, B$
20 AC = ASC(A$)
30 BC = ASC(B$)
40 IF AC = BC THEN PRINT "SONO UGUALI"
50 IF AC > BC THEN PRINT A$; " È DOPO "; B$
60 IF AC < BC THEN PRINT A$; " È PRIMA DI "; B$
```

Spiegazione:

- Converte i caratteri in codice numerico ASCII per confrontarli

Esercizio: Estendi per confrontare intere stringhe (con MID$ e ciclo FOR)

Livello 3 – Controllo I/O e Hardware ESP32

ESEMPIO 12 — Lampeggio LED su un pin

Comandi usati: PINMODE, DIGITALWRITE, DELAY

```
10 PINMODE(2, 1, 0)     ' Pin 2 come OUTPUT
20 DIGITALWRITE(2, 1)   ' Accende LED
30 DELAY 500
40 DIGITALWRITE(2, 0)   ' Spegne LED
50 DELAY 500
60 GOTO 20
```

Spiegazione:

- Imposta il pin come uscita
- Accende e spegne il LED ogni 0.5 secondi in loop

Esercizio: Prova con DELAYMILLIS + TI per un blink non bloccante.

ESEMPIO 13 — Pulsante: premi per accendere

Comandi usati: PINMODE, DIGITALREAD, DIGITALWRITE

```
10 PINMODE(2, 1, 0)   ' OUTPUT LED
```

```
20 PINMODE(4, 0, 1)   ' INPUT con pull-up su pin 4
30 B = DIGITALREAD(4)
40 IF B = 0 THEN DIGITALWRITE(2, 1) ELSE DIGITALWRITE(2, 0)
50 DELAY 100
60 GOTO 30
```

Spiegazione:

- Se premi il pulsante (connesso a GND), il LED si accende
- Usa logica con IF e lettura del pin

Esercizio: Fai lampeggiare il LED finché il pulsante è premuto.

ESEMPIO 14 — Lettura analogica da sensore

Comandi usati: ANALOGREAD, PRINT, DELAY

```
10 PRINT "LETTURA ANALOGICA:"
20 V = ANALOGREAD(36)   ' ADC1_0 = GPIO36 su ESP32
30 PRINT "VALORE = "; V
40 DELAY 500
50 GOTO 20
```

Spiegazione:

- Legge da un sensore (es. potenziometro) su pin ADC
- Valori vanno da 0 a 4095

Esercizio: Visualizza una barra con SPC() proporzionale al valore.

ESEMPIO 15 — Semaforo semiautomatico

Comandi usati: DIGITALWRITE, PINMODE, DELAY, FOR

```
10 PINMODE(2, 1, 0)   ' VERDE
20 PINMODE(4, 1, 0)   ' GIALLO
30 PINMODE(5, 1, 0)   ' ROSSO
40 FOR I = 1 TO 3
50   DIGITALWRITE(2, 1): DELAY 3000: DIGITALWRITE(2, 0)
60   DIGITALWRITE(4, 1): DELAY 1000: DIGITALWRITE(4, 0)
70   DIGITALWRITE(5, 1): DELAY 3000: DIGITALWRITE(5, 0)
80 NEXT I
90 PRINT "FINE CICLO"
```

Spiegazione:

- Simula un semaforo con 3 LED
- Ogni colore ha durata definita e viene ripetuto 3 volte

Esercizio: Aggiungi un input per attivare un ciclo d’emergenza lampeggiante.

ESEMPIO 16 — Blink LED con DELAYMILLIS e TI

Comandi usati: PINMODE, DIGITALWRITE, DELAYMILLIS, TI

```
10 PINMODE(2, 1, 0)
20 DELAYMILLIS 500
30 T = TI
40 STATO = 0
50 WHILE 1
60   IF TI >= T + 50 THEN
70     STATO = NOT STATO
80     DIGITALWRITE(2, STATO)
90     T = TI
100  END IF
110 WEND
```

Spiegazione:

- Lampeggia il LED senza bloccare il programma
- Usa TI e DELAYMILLIS per una pausa gestita

Esercizio: Aggiungi controllo con un pulsante per fermare il lampeggio.

Livello 4 – Gestione File e Memoria

ESEMPIO 17 — Salvataggio e caricamento programma

Comandi usati: INPUT, SAVE, NEW, LOAD, RUN

```
10 INPUT "NOME DEL FILE: "; F$
20 SAVE F$
30 PRINT "PROGRAMMA SALVATO IN "; F$
40 NEW
50 LOAD F$
60 RUN
```

Spiegazione:

- Salva il programma attuale con nome a scelta
- Cancella dalla memoria (NEW) e lo ricarica (LOAD)
- Lo esegue subito

Esercizio: Prova a salvare su memoria interna con SAVEINT.

ESEMPIO 18 — Rubrica con salvataggio persistente

Comandi usati: DIM, INPUT, SAVEINT, LOADINT, PRINT

```
10 DIM NOMI$(3), NUM$(3)
20 FOR I = 1 TO 3
30   INPUT "NOME "; NOMI$(I)
40   INPUT "NUMERO "; NUM$(I)
50 NEXT I
60 SAVEINT "rubrica.bas"
70 PRINT "SALVATA. ORA LA RICARICO:"
80 LOADINT "rubrica.bas"
90 FOR I = 1 TO 3
100  PRINT NOMI$(I); " - "; NUM$(I)
110 NEXT I
```

Spiegazione:

- Rubrica base salvata su memoria interna
- Dopo LOADINT, i dati sono di nuovo disponibili

Esercizio: Estendi per 10 contatti. Aggiungi DIRINT per mostrare i file salvati.

ESEMPIO 19 — Esplora i file disponibili

Comandi usati: DIR, DIRINT, PRINT

```
10 PRINT "FILE SU SCHEDA SD:"
20 DIR
30 PRINT "FILE SU MEMORIA INTERNA:"
40 DIRINT
```

Spiegazione:

- Mostra l’elenco dei file disponibili nei due spazi di archiviazione

Esercizio: Dopo ogni DIR, aggiungi una pausa INPUT "PREMI INVIO PER CONTINUARE".

ESEMPIO 20 — Verifica integrità di un file

Comandi usati: VERIFY, VERIFYINT

```
10 INPUT "FILE DA VERIFICARE: "; F$
20 VERIFY F$
30 VERIFYINT F$
```

Spiegazione:

- Confronta file attuale in memoria con una copia salvata
- Utile per capire se ci sono state modifiche

Esercizio: Usa IF VERIFY con un messaggio di "modificato/senza modifiche".

ESEMPIO 21 — Log automatico con orario

Comandi usati: STR$, TI$, PRINT, SAVEINT

```
10 T$ = TI$
20 M$ = "AVVIO PROGRAMMA ALLE: " + T$
30 PRINT M$
40 SAVEINT "log_" + LEFT$(T$, 4) + ".txt"
```

Spiegazione:

- Registra in un file il momento dell’avvio
- Il file prende nome da TI$, utile per identificare log cronologici

Esercizio: Aggiungi una seconda riga con "FINE PROGRAMMA" a distanza di 5 secondi.

Livello 5 – Funzioni e Progetti Avanzati

ESEMPIO 22 — Definisci una funzione quadrato

Comandi usati: DEF FN, PRINT, FOR

```
10 DEF FN Q(X) = X * X
20 FOR I = 1 TO 5
30 PRINT "QUADRATO DI "; I; " = "; FN Q(I)
40 NEXT I
```

Spiegazione:

- Definisce una funzione personalizzata Q che restituisce il quadrato di un numero
- Poi la usa in un ciclo

Esercizio: Crea una funzione CUBO(X).

ESEMPIO 23 — Conta le vocali in una stringa

Comandi usati: INPUT, LEN, MID$, IF, FOR

```
10 INPUT "INSERISCI UNA PAROLA: "; S$
20 C = 0
30 FOR I = 1 TO LEN(S$)
40   L$ = MID$(S$, I, 1)
50   IF L$ = "A" OR L$ = "E" OR L$ = "I" OR L$ = "O" OR L$ = "U" THEN C = C + 1
60 NEXT I
70 PRINT "VOCALI TROVATE: "; C
```

Spiegazione:

- Analizza una stringa, estrae ogni lettera, verifica se è una vocale

Esercizio: Rendi il confronto **case-insensitive**.

ESEMPIO 24 — Slot machine testuale

Comandi usati: RND, PRINT, FOR, INPUT, IF

```
10 SYMBOLS$ = "*#@"
20 INPUT "PREMI INVIO PER GIRARE: "; W$
30 FOR I = 1 TO 3
40   X = INT(RND(3)) + 1
50   PRINT MID$(SYMBOLS$, X, 1);
60 NEXT I
70 PRINT
80 GOTO 20
```

Spiegazione:

- Estrae casualmente 3 simboli da una stringa
- Simula una slot machine a riga singola

Esercizio: Aggiungi punteggio se escono 3 simboli uguali.

ESEMPIO 25 — Visualizzatore orario dinamico

Comandi usati: CLS, TI$, DELAY, PRINT

```
10 CLS
20 FOR I = 1 TO 10
30   PRINT "ORARIO: "; TI$
40   DELAY 1000
50   CLS
60 NEXT I
```

Spiegazione:

- Mostra un orologio aggiornato ogni secondo per 10 volte

Esercizio: Aggiungi una barra di avanzamento visuale con SPC(I).

ESEMPIO 26 — Simulatore casuale "Test della fortuna"

Comandi usati: RND, IF, INPUT, PRINT

```
10 INPUT "VUOI TENTARE LA FORTUNA? (S/N): "; R$
20 IF R$ <> "S" THEN END
30 N = RND(10)
40 IF N = 7 THEN PRINT "FORTUNA INCREDIBILE!"
50 IF N > 4 AND N < 7 THEN PRINT "NI... CI SEI ANDATO VICINO"
60 IF N < 5 THEN PRINT "NON QUESTA VOLTA!"
70 GOTO 10
```

Spiegazione:

- Estrae un numero da 0 a 9
- Valuta tre fasce di risposta

Esercizio: Tieni il punteggio positivo e negativo con due variabili.

ESEMPIO 27 — Mini editor da terminale (base)

Comandi usati: INPUT, SAVEINT, PRINT, GOTO, IF, LET

```
10 INPUT "NOME FILE: "; F$
20 PRINT "SCRIVI TESTO. DIGITA SOLO 'FINE' PER TERMINARE."
30 T$ = ""
40 INPUT ""; RIGA$
50 IF RIGA$ = "FINE" THEN GOTO 80
60 T$ = T$ + RIGA$ + CHR$(13)
70 GOTO 40
80 PRINT "SALVO IN "; F$
90 SAVEINT F$
```

Spiegazione:

- Raccoglie righe da tastiera
- Le concatena in una stringa
- Salva tutto in un file con nome personalizzato

Esercizio: Aggiungi data e orario all'inizio con TI$.

INDICE