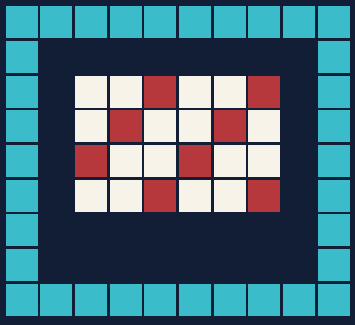
NDB SOFTWARE

NES BASIC PROGRAMMING GUIDE

Small computer. Real programs.



8-BIT IMPOSTOR

Edition 1.0 / 3 November 2026
Offline programming book / NTSC language edition

CONTENTS

Choose a chapter in this linked contents list or use your reader's bookmarks.

1. Welcome to NES BASIC 5

2. Quick start: type, run, keep 7

3. The editor and its keyboards 9

4. Language fundamentals 11

5. Strings and text values 13

6. Input and output 15

7. Decisions, loops and subroutines 17

8. Functions you can rely on 19

9. Graphics and the built-in sprite creator 21

10. Sound and short melodies 23

11. Controllers, input events and a game loop 25

12. Save and load without losing work 27

13. Three complete projects to make your own 29

14. Alphabetical command and function reference 31

15. Errors, recovery and troubleshooting 60

16. Appendices: codes, limits and edition history 64

ABOUT THIS EDITION

Edition 1.0 · 3 November 2026 · NDB Software

Compatibility: 8-Bit Impostor 0.7.1 (30), with the completed native NES BASIC ROM. Language fingerprint: `2b2901a4845428190b89d1ae8ccfb2b05ffb18195971d1f134f0ebb5098b2c1d`. The cartridge's V1.0 welcome banner is not enough to identify its features. Older imported ROMs may have a smaller language. All examples in this edition are checked against the fingerprint above. Physical NES tests are NOT VERIFIED.

1. Welcome to NES BASIC

A small computer can be a good place to learn programming. You type a line, run it, and see exactly what changes. NES BASIC turns an NES cartridge into that kind of computer: a language, a text editor, a graphics canvas, hardware sprites, sound and a small project library, all running on the emulated NES processor.

8-Bit Impostor provides the host screen, touch controls, controller connections, optional external-keyboard transport and persistent battery files. It does not evaluate BASIC for the cartridge. The same standalone ROM has its own controller keyboard and interpreter. The PDF you are reading is a host document, not an NES program; a physical NES does not display this book.

This is a deliberately small integer BASIC. There are single-letter numeric variables and independent single-letter string variables. There are no arrays, floating-point numbers, ELSE, DATA/READ or ON GOTO. Its limitations are useful to know early: they make your programs easier to plan and your errors easier to fix. The feature matrix in the appendices describes this edition rather than another BASIC dialect.

BASIC, NES PAINT, EDITOR, CALCULATOR, NES SHEETS, BLOCKDROP and ASTRO BLASTER are separate cartridges in NDB SOFTWARE. Opening BASIC does not depend on opening Paint, and their saved projects are not shared. This guide concentrates on BASIC.

You can keep a finished image or sprite visible when a program ends. You can also interrupt a loop, return to editing and run it again. Use explicit SAVE to keep source and artwork before resetting or leaving. Saving a program is different from saving an emulator snapshot; Chapter 12 explains both.

2. Quick start: type, run, keep

Open NDB SOFTWARE, choose NES BASIC, then PLAY / RESUME. The cartridge prints its welcome and READY. Press Select to open the on-screen keyboard. Move with the D-pad, insert with A, backspace with B and submit with Start. On an external keyboard, type normally and use Enter. Submit each numbered line separately.

```
10 PRINT "HELLO FROM NES BASIC"
20 END
```

Now submit `RUN`. The output is `HELLO FROM NES BASIC`. Submit `LIST` to see the two stored lines. A number at the start stores a line; an unnumbered submission is a direct command. The number is an address in your program, not part of the printed message. Leaving gaps of ten makes it easy to insert lines later.

To change the message, resubmit line 10 with a different quoted string. To delete line 20, submit `20` by itself. The editor is a line editor, not a full-screen source editor: LIST displays stored code, and you replace a stored line by typing its number and new contents. Source is kept sorted by line number.

Submit `SAVE "HELLO"` and wait for `SAVED "HELLO"`. After a cold start, use `LOAD "HELLO"`, then `RUN`. `RUN "HELLO"` loads and runs the named file in one step. If a confirmation appears, A/Y/Enter accepts; B/N/Select/Escape cancels. A saved file includes your program and all kept BASIC artwork.

SAVE before reset or leaving. An ordinary RUN does not save your program.

READY is printed at startup. Successful ordinary commands return quietly to the editing cursor; the lack of another READY message is normal. Errors show a message and return to editing. For a runtime error, `IN 100` identifies the source line where execution failed. Correct that line and try RUN again.

3. The editor and its keyboards

The native keyboard has letter and symbol pages. SHFT changes pages; SPCE inserts a space. Select opens or closes it without submitting the current line. With the keyboard open, the D-pad chooses keys. With it closed, Left/Right moves the caret through the current line. Reopen it to insert at that position. B removes the character before the caret. Insertion moves the rest of the line to the right.

The input line accepts at most 95 characters and wraps on the 28-column editing area. That physical input limit is separate from the tokenized program capacity. A stored line also needs its line number and record metadata. A string variable has a smaller, separate limit of 63 bytes; do not confuse these limits.

External keyboard input follows the same ROM editor. Enter submits, Backspace deletes, Escape closes the controller keyboard, Left/Right moves the caret, and Cmd+Left/Right goes to the start/end. Ctrl+C interrupts a running program. Source entry normalizes lowercase letters to uppercase, including characters entered inside string literals. There is no Unicode source editor or general paste-file importer for .bas documents; enter or type the listings as separate submissions.

The optional external-keyboard bridge uses virtual controller port two. After its handshake, JOY(2) returns zero until reset. It is not the original Family BASIC keyboard protocol. You can always use the on-screen keyboard with ordinary NES

controller input, including in another compatible emulator.

NEW clears working source and variables after confirming a nonempty dirty program. It keeps your kept artwork. CLEAR resets values and control state while keeping source and artwork. RUN starts from the beginning and resets variables. Use LIST to review a program before replacing it. Long LIST/output pauses after 18 rows: Start continues and Select cancels with BREAK.

4. Language fundamentals

Numbers are checked signed 16-bit integers: -32768 through 32767. A division truncates toward zero, so -7/3 is -2. Overflow raises OVERFLOW rather than silently wrapping. Dividing by zero raises DIVISION BY ZERO. Integer INT is an identity operation; the language has no decimal fraction type to round.

There are 26 numeric variables, A through Z. Assignment can be written `A=42` or `LET A=42`. A variable defaults to zero. Multi-letter names such as TOTAL and names with digits such as X1 are unsupported. Choose a letter and use REM to say what it means. String A$ is a separate variable from numeric A.

Expressions support unary +/-, multiplication/division, addition/subtraction, parentheses and comparisons. Precedence is unary, then multiply/divide, then add/subtract or string concatenation, then comparisons. For example, 2+3*4 is 14; (2+3)*4 is 20. Comparisons `= <> < > <= >=` produce numeric 1 for true and 0 for false. There are no AND/OR/NOT or bitwise operators in this edition.

A colon separates statements on the same source line. `REM` ignores the remainder of its line, including colons and words that would otherwise be keywords. Use comments for intent and units. Keywords inside a quoted string are text. Unclosed double quotes raise SYNTAX ERROR. Extra punctuation is not ignored.

Line numbers range from 1 to 65535, independently of the signed numeric value range. A program holds at most 128 lines and 3072 tokenized source bytes. Keywords are compact tokens, so typed character count is not the same as stored byte count. The expression parser has a guarded nesting limit, and FOR/GOSUB share an eight-frame control stack. Keep expressions and nesting modest.

Failed line storage does not replace a previously valid line. Fix the reported

syntax or memory error rather than resetting the whole cartridge.

5. Strings and text values

A$ through Z$ are 26 string variables, independent of A through Z. They default to empty strings. String assignment, copying, concatenation and comparison are real typed operations. `A$="HELLO"`, `B$=A$` and `C$=A$+"!"` are valid direct fragments. The + operator concatenates strings only when both operands are strings.

Numeric and string values do not convert silently. `A$=42`, `A=A$` or `A$+1` raises TYPE MISMATCH. Use STR$ to make decimal text from an integer, or VAL to parse a whole decimal string. This makes a program's intended conversion visible.

An empty literal is `""`. To include a quotation mark, double it inside a literal: `A$="SAY ""HI"""` stores SAY "HI". String INPUT treats typed spaces and quotation marks as data rather than source syntax. Empty string input is valid.

All six comparison operators work between strings. Comparison is lexicographic on unsigned byte codes: an equal prefix sorts before a longer string. Thus "A" sorts before "AA". Comparison gives numeric 1/0. A string cannot be compared to a number without an explicit conversion. The uppercase-oriented font is not a Unicode collation system.

A stored or concatenated string can contain at most 63 bytes. All 26 variables share a compacted 256-byte pool. Empty strings use no pool bytes. Expressions also use a bounded 128-byte temporary arena. STRING TOO LONG means a result exceeds the length limit; STRING SPACE FULL means shared/temporary capacity is exhausted. The old destination

survives a failed assignment. Shorten or clear values, or use CLEAR when it is safe to reset all values.

```
10 INPUT "YOUR NAME";N$
20 A$="HELLO, "+N$
30 PRINT A$
40 IF N$="" THEN PRINT "EMPTY NAME"
50 END
```

RUN, enter NES USER and submit. The result is HELLO, NES USER. RUN again and submit an empty answer: the program prints HELLO, followed by EMPTY NAME. The empty test compares N$ to an empty string, not to numeric zero. This listing is also the first complete project in Chapter 13.

6. Input and output

PRINT accepts numeric/string expressions and literal text. A semicolon places the next value immediately after the previous one. A comma advances to the next four-column tab stop. With no trailing separator, PRINT ends its output line. A trailing separator leaves the output position for the next PRINT. Comma is a formatting separator here; do not assume another BASIC dialect's larger zones.

`PRINT "VALUE=";A` prints the label and the number together. `PRINT A;" ";A$` prints the independent numeric and string values with a space. These are direct fragments rather than complete stored programs. PRINT output has a bounded buffer; OUTPUT TOO LONG is recoverable.

INPUT accepts a numeric variable or a string variable, with an optional prompt expression and semicolon: `INPUT "AGE";A` or `INPUT "NAME";N$`. The cartridge adds its question marker. Numeric input must be a valid signed integer; an empty or invalid answer raises BAD INPUT and retries. String input preserves spaces and accepts empty input; overlong or pool-full entries retry without changing the old destination. B/Backspace edits the answer. Ctrl+C or the emergency pad gesture cancels and recovers.

`LOCATE column,row` positions text output at column 0–27, row 0–21. `CLS` clears the active text terminal or graphics canvas. PRINT, INPUT and LOCATE restore text when used from graphics, so they can remove a graphics image. Keep a program's graphics section and text reporting section distinct

when that matters.

INKEY$ reads one newly queued input event without waiting; INPUT collects an edited whole answer and can wait. They serve different jobs. A loop using INKEY$ must test for an empty result before calling ASC. The input-code project in Chapter 11 shows the pattern.

7. Decisions, loops and subroutines

IF evaluates a numeric condition. A nonzero value is true. THEN can name a literal line target or start an inline statement. In a running program, `IF A=1 THEN 100` jumps to line 100. `IF A=1 THEN PRINT "YES"` executes the inline PRINT. There is no ELSE. A false IF skips the entire remainder of its line, including colon-separated statements; put work that must always run on a different source line.

GOTO jumps to a literal line number. The target must exist. GOSUB jumps and saves a return location; RETURN resumes after that call, including a colon continuation. A missing target reports UNDEFINED LINE. RETURN without a matching call, or natural end of source while a GOSUB is still pending, reports INVALID RETURN. END/STOP intentionally terminate the program and discard pending control frames.

```
10 GOSUB 100
20 PRINT "BACK"
30 END
100 PRINT "IN SUBROUTINE"
110 RETURN
```

This prints IN SUBROUTINE, then BACK. The END prevents normal execution from falling into the subroutine. A subroutine is a convention using line numbers, not a separately declared function with arguments or local variables.

FOR/NEXT repeats using a numeric variable, inclusive limit and optional nonzero STEP. Positive and negative steps work. A loop whose initial value is already past its limit performs zero trips. NEXT names the same loop variable. Do not reuse an active loop variable for another FOR. A missing or mismatched NEXT reports UNMATCHED LOOP. FOR/GOSUB nesting together can use eight frames.

```
10 A=0
20 FOR I=1 TO 5
30 A=A+I
40 NEXT I
50 PRINT "TOTAL=";A
60 END
```

This prints TOTAL=15. After the loop, I is 6. Changing the step changes which values are added. Arithmetic that overflows while advancing a loop still raises OVERFLOW. To stop a runaway program, hold Start+Select for 60 emulated frames (about a second at normal NTSC speed), or use external-keyboard Ctrl+C.

8. Functions you can rely on

ABS(N) returns absolute value; ABS(-32768) overflows. INT(N) returns the same integer. SGN(N) returns -1, 0 or 1. RND(N) requires N from 1 to 32767 and returns 0 through N-1. The ROM's deterministic 16-bit generator starts from seed ACE1 when values reset. RUN repeats that initial sequence; there is no host-random number dependency or RANDOMIZE statement. Use RND(6)+1 for values 1–6.

ASC(S$) returns the first byte of a nonempty string. ASC("A") is 65, ASC("0") is 48 and ASC(" ") is 32. ASC("") raises BAD ARGUMENT. CHR$(N) creates one byte for N=0–255, including zero. Every value in that domain round-trips through ASC(CHR$(N)). A valid byte is not necessarily a visible glyph; reserved button bytes and control bytes are data, not printable ASCII. CHR$ does not discover hardware key codes.

LEN counts string bytes, including embedded NUL. LEFT$/RIGHT$ take a nonnegative count and clamp it to available bytes. MID$ uses a positive one-based start and optional nonnegative count; omitted count means the rest. A start past the end returns empty. Negative counts or start zero raise BAD ARGUMENT.

STR$ produces ordinary signed decimal without a leading space for positive numbers. VAL accepts a whole signed decimal with optional surrounding spaces. VAL("12X"), VAL("") and embedded NUL are invalid; out-of-range values overflow. These are not floating-point conversion routines. Function arguments retain their types; LEN(42), for example, raises

TYPE MISMATCH.

INKEY$ has no parentheses. It returns one queued byte or an empty string immediately. It reads events while a program runs; a direct idle read normally returns empty. The controller-event table and focus rules are in Chapter 11. JOY(port) returns held-state masks and SPRCOL(a,b) returns a box-collision result; those numeric functions are described with sprites below and in the reference.

9. Graphics and the built-in sprite creator

SCREEN 0 is text. SCREEN 1 starts/clears a graphics canvas and its standard mapping. The canvas is 128×64 one-bit pixels, with local (0,0) at its top left and (127,63) at its bottom right. It occupies part of the NES screen, not a full-screen bitmap. COLOR selects one shared foreground/background pair. Valid NES palette indices are 0–63 except 13,29,45,61. The canvas does not have a separate color per pixel.

```
10 SCREEN 1
20 COLOR 42,15
30 RECT 8,8,119,55
40 LINE 0,0,127,63
50 PSET 64,32
60 END
```

PSET plots one pixel. LINE is an inclusive integer segment. RECT uses opposite corners, not width and height. An optional final ink argument of 0/1 clears/sets pixels. Points outside the canvas clip. FILL fills a four-connected region with a bounded worklist; a complicated region can report FILL TOO COMPLEX. Large operations paint progressively and remain interruptible.

TILE works on the NES nametable in SCREEN 1: column 0–31, row 0–29, pattern 0–255. Patterns are shared. A TILE referencing a canvas pattern changes when that pattern is drawn later. SCREEN 1 restores the standard mapping. CLS clears the active canvas; placements outside it remain. Text

restoration clears the canvas.

Sprites are real 8×8 NES hardware sprites. SPRITE slot,x,y,art shows one of 64 slots, using x=0–255 and y=1–239. Art 0–7 is predefined: ship, player, enemy, ball, projectile, star, block, pointer. Art 8–39 is editable. MOVE changes a shown slot's position; HIDE hides it. SPRCOL(a,b) tests unclipped 8×8 bounding boxes of visible slots, including transparent pixels. It is not per-pixel collision. The ninth sprite on the same scanline is hidden by the hardware limit; no automatic flicker is added. Arrange sprites accordingly.

EDITSPR 8 opens BASIC's own creator, independent of Paint. D-pad moves/wraps on the grid, A paints and B erases. Select opens tools; choose color/art with the pad. Tools also clear, fill, mirror horizontally, flip vertically and undo. Start requests exit. If edited, choose KEEP, DISCARD or CANCEL. Kept art becomes available to SPRITE immediately; it is not permanent until you SAVE the project.

Each pattern has four indexed pixel values: 0 transparent, 1 green, 2 white, 3 blue. This is a shared fixed sprite palette, not arbitrary per-sprite color editing. There is no composed 16×16 editor or independently named sprite file. A named BASIC project stores all 32 kept 8×8 patterns. Reopen the art ID to edit it. After KEEP, use SCREEN 1:SPRITE 0,80,80,8 as a direct fragment to display art 8.

10. Sound and short melodies

BEEP plays a short note on pulse channel one. SOUND channel,pitch,duration starts or replaces a voice asynchronously. Channels 1/2 are pulse, 3 triangle, 4 noise. Tonal notes are integers 33–96 using MIDI note numbering as a convention: 60 is C4, 64 E4, 67 G4 and 69 A4. No MIDI device is required. Noise pitch is a hardware period index 0–15 rather than a musical note.

Duration is 0–32767 emulated frames. Zero mutes that voice, but its channel and pitch must still be valid. SOUND followed by WAIT makes notes sequential. Other BASIC statements add their own frame cost, so this is a simple sequencer, not an exact wall-clock audio workstation. The target is NTSC; PAL retuning is not part of this language edition.

```
10 SOUND 1,60,20
20 WAIT 20
30 SOUND 1,64,20
40 WAIT 20
50 SOUND 1,67,30
60 WAIT 30
70 END
```

This plays three notes and ends after their waits. Separate channels can form a chord. Voices have fixed amplitudes; there is no volume argument, music-string language or DMC/sample playback API. Normal source completion lets finite tones finish; END/STOP, errors, break, NEW/RUN and sprite-editor entry silence them. Pausing freezes emulation and can preserve a remaining note until resume.

If no sound is heard, check the device's silent mode, audio route, system volume and the app's VIDEO / AUDIO settings. A change of audio route can pause the app. Do not diagnose a silent phone as a missing BASIC command without checking those.

11. Controllers, input events and a game loop

JOY(1) samples held virtual NES buttons. The mask is A=1, B=2, Select=4, Start=8, Up=16, Down=32, Left=64, Right=128. Simultaneous buttons add. Read once into J if several tests should use the same sample. There is no bitwise AND, but integer arithmetic extracts a bit: J/128 is Right; J/16-J/32*2 is Up.

INKEY$ instead queues new press edges on port one. Codes are A=128, B=129, Select=130, Start=131, Up=132, Down=133, Left=134, Right=135. They are reserved codes, not ASCII. Held gameplay buttons do not repeat each frame. Simultaneous edges queue in that order. Eight events fit; overflow drops the newest and increments a saturating internal counter. RUN/focus/input transitions clear the queue; the Start that launches RUN is not also an input event.

```
10 K$=INKEY$
20 IF K$="" THEN GOTO 10
30 PRINT "CODE=";ASC(K$)
40 GOTO 10
```

Press A while it runs: CODE=128 appears. An idle empty result loops before ASC, avoiding ASC(""). External keyboard text during RUN queues text bytes; it does not discover arbitrary USB/Bluetooth scan codes. The host or adapter maps a physical device to virtual NES buttons. Compatibility depends on that mapping.

Hold Select alone for 30 frames during RUN to open text keys. D-pad navigation then selects text rather than sending game-direction events. A sends the chosen character, B sends byte 8, Start sends byte 13, Select closes the keys. Short Select remains event 130. Start+Select is reserved for the emergency break. INPUT owns a separate edited-answer workflow, so do not expect it to consume these queued game events. After external-keyboard handshake, JOY(2) is zero.

```
10 SCREEN 1:X=80
20 SPRITE 0,X,100,0
30 J=JOY(1)
40 X=80+J/128*8
50 MOVE 0,X,100
60 WAIT 1:GOTO 30
```

Hold Right: the ship moves from X=80 to X=88. Release Right: it returns to 80. This small complete game loop demonstrates sampling, changing state, MOVE and WAIT without leaving the valid coordinate range. It does not implement continuous acceleration. You can replace art 0 with an art ID you created, then save the modified source and artwork. Chapter 13 suggests safe extensions.

12. Save and load without losing work

BASIC has six named project slots. SAVE "NAME" stores source and all 512 bytes of kept editable artwork; it does not store variable values, the canvas, active sound or execution position. Names contain 1–12 uppercase letters/digits or underscore, hyphen and period. SAVE or LOAD without a name uses PROGRAM. Wait for the success notice before leaving the cartridge.

SAVE asks before overwriting an existing name. LOAD asks before discarding dirty working code/art. A/Y/Enter accepts; B/N/Select/Escape cancels. FILES and CAT list the library. KILL "NAME" or ERASE "NAME" deletes after confirmation; KILL 1 (or ERASE 1) can explicitly remove a numbered slot, including an unreadable record. A full directory reports SAVE DIRECTORY FULL; overwrite intentionally or erase a file you no longer need. Slots are not automatically reclaimed from corruption.

LOAD validates a saved record before replacing working source/art. An intact name with damaged metadata/payload reports BAD SAVED PROJECT. A damaged name can produce FILE NOT FOUND instead. Do not repeatedly overwrite your only backup while troubleshooting. A checksummed native old-record journal recovers interrupted writes at boot or the next storage command when that backup is valid. It cannot repair arbitrary electrical loss or a corrupted backup.

New records are version 3; this ROM reads versions 2 and 3 without silently rewriting older files. An older version-2 ROM rejects version-3 records. The app has scoped same-title battery migration for reviewed predecessors, keeping the old file and preferring an existing current save. Arbitrary changed ROMs are not automatically interchangeable. Keep the original ROM/save when archiving work.

The host's SAVE STATES is a separate emulator feature: snapshots belong to an exact ROM hash, core version and region. BASIC has no automatic-resume promise; use its explicit project commands. Loading an emulator snapshot keeps current committed cartridge files, rather than rolling those files back. These two forms of saving solve different problems.

On a physical board, SAVE requires compatible MMC1/SXROM banking and retained 32 KiB PRG SRAM, plus 8 KiB writable CHR RAM. A normal ROM chip alone cannot save; an 8 KiB SNROM board is insufficient. Emulator battery-file restart tests are verified, but physical NES, flashcart model, PCB/programmer and power-removal retention tests remain NOT VERIFIED.

13. Three complete projects to make your own

Project 1 is the name greeter in Chapter 5. Type its exact five lines and RUN. Try both NES USER and an empty answer. N$ holds input, A$ holds the greeting, + joins text, and the final IF notices empty input. Add a second independent numeric variable and PRINT it; never assign a number directly to A$. Save the finished version as GREETER. After a cold start, LOAD "GREETER" and RUN.

Project 2 is the sum in Chapter 7. Its output TOTAL=15 is an easy check of the loop. Change the limit from 5 to 10 and reason about the result before running. Try a descending loop with a negative STEP. Keep arithmetic within -32768...32767. A REM comment on a separate line can explain which variable is the running total. Save a working version as SUM before experimenting further.

Project 3 is the ship controller in Chapter 11. Its expected positions are 80 and 88. Draw art 8 in EDITSPR, KEEP it, then change line 20's art argument from 0 to 8. Save the project as SHIP. Load it after reset to check source and artwork together. Keep MOVE coordinates in range if you extend it. For smooth motion, add separate held-bit tests and explicit bounds before updating X; the original simple listing deliberately avoids accidental coordinate overflow.

The frame/diagonal and melody listings are additional complete samples. The input code diagnostic is a useful tool when you change controller mappings. All eight complete

listings in this book have matching .bas source fixtures and executed ROM evidence. Short code fragments in prose/reference illustrate syntax and are not claimed to be standalone numbered programs.

14. Alphabetical command and function reference

Notation: N is a numeric expression; S$ a string expression; A a numeric variable; A$ a string variable; brackets mean an optional part and are not typed. Statements have no expression return value. A line target is a literal line number. A source line can contain colon-separated statements unless a command is direct-only. Functions return a typed value and are used inside expressions. THEN/TO/STEP are syntax fragments used by their parent statements. The reference covers all 59 shipped keyword tokens; it does not borrow commands from another BASIC dialect.

ABS

Syntax: `ABS(N)`

Parameters: N: signed integer.

Result: integer.

Example fragment: `PRINT ABS(-5)`

Absolute value; -32768 has no positive signed 16-bit representation.

Errors: OVERFLOW (9); TYPE MISMATCH (25).

ASC

Syntax: `ASC(S$)`

Parameters: S$: nonempty string.

Result: integer.

Example fragment: `PRINT ASC("A")`

First unsigned byte,0–255; does not inspect arbitrary device scan codes.

Errors: BAD ARGUMENT (11) for empty; TYPE MISMATCH (25).

BEEP

Syntax: `BEEP`

Parameters: No arguments.

Result: statement; no value.

Example fragment: `BEEP`

Starts/replaces pulse 1 note 69 for 12 emulated frames. Other voices remain.

Errors: SYNTAX ERROR (1) for arguments.

CAT

Syntax: `CAT`

Parameters: No arguments; direct command.

Result: statement; no value.

Example fragment: `CAT`

Alias for FILES. Lists six project slots; does not LOAD a project.

Errors: SYNTAX ERROR (1); DIRECT COMMAND ONLY (6).

CHR$

Syntax: `CHR$(N)`

Parameters: N: integer 0–255.

Result: string.

Example fragment: `A$=CHR$(72)+CHR$(73)`

Creates exactly one byte, including NUL. A supported byte need not be a printable glyph.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25); STRING SPACE FULL (26).

CLEAR

Syntax: `CLEAR`

Parameters: No arguments.

Result: statement; no value.

Example fragment: `CLEAR`

Resets numeric/string values, control frames and random seed; stops sound/hides sprites. Keeps source and kept artwork.

Errors: SYNTAX ERROR (1).

CLS

Syntax: `CLS`

Parameters: No arguments.

Result: statement; no value.

Example fragment: `CLS`

Clears terminal in text mode, canvas in graphics. Does not remove saved source or all hardware sprites.

Errors: SYNTAX ERROR (1).

COLOR

Syntax: `COLOR foreground,background`

Parameters: Two numeric NES palette indices 0–63, excluding 13/29/45/61.

Result: statement; no value.

Example fragment: `COLOR 42,15`

One shared foreground/background pair. Text may also use it; sprite palette remains fixed.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

DEMOS

Syntax: `DEMOS`

Parameters: No arguments; direct command.

Result: statement; no value.

Example fragment: `DEMOS`

Lists six editable built-in examples. Does not use saved-file slots.

Errors: SYNTAX ERROR (1); DIRECT COMMAND ONLY (6).

EDITSPR

Syntax: `EDITSPR art`

Parameters: Numeric art ID 8–39; whole direct command.

Result: statement; no value.

Example fragment: `EDITSPR 8`

32 native 8×8 two-bit patterns. Keep/discard/cancel; KEEP is session-only until project SAVE. No colon continuation.

Errors: BAD ARGUMENT (11); DIRECT COMMAND ONLY (6); SYNTAX ERROR (1).

END

Syntax: `END`

Parameters: No arguments.

Result: statement; no value.

Example fragment: `30 END`

Finishes execution, clears pending control frames and stops sound. Keeps source and final scene.

Errors: SYNTAX ERROR (1).

ERASE

Syntax: `ERASE ["NAME" | slot]`

Parameters: Direct command; name rules match SAVE, slot 1–6.

Result: statement; no value.

Example fragment: `ERASE "DEMO"`

Confirmed deletion, alias of KILL; cancel retains the file. Without a name uses PROGRAM.

Errors: FILE NOT FOUND (22); BAD ARGUMENT (11); DIRECT COMMAND ONLY (6).

FILES

Syntax: `FILES`

Parameters: No arguments; direct command.

Result: statement; no value.

Example fragment: `FILES`

Lists the six-slot directory, including available/damaged entries. Does not clear working code.

Errors: SYNTAX ERROR (1); DIRECT COMMAND ONLY (6).

FILL

Syntax: `FILL x,y[,ink]`

Parameters: Numeric seed in local canvas coordinates; ink 0/1, default 1.

Result: statement; no value.

Example fragment: `FILL 20,20`

SCREEN 1 required. Four-connected bounded flood fill; clipped/already-matching seed is a no-op.

Errors: FILL TOO COMPLEX (21); GRAPHICS MODE REQUIRED (20); BAD ARGUMENT (11); TYPE MISMATCH (25).

FOR

Syntax: `FOR A=start TO limit [STEP step]`

Parameters: Numeric single-letter variable; numeric expressions; nonzero step, default 1.

Result: statement; no value.

Example fragment: `10 FOR I=1 TO 5`

Program loop, inclusive limit; negative/zero-trip loops supported. Shares eight-frame stack with GOSUB.

Errors: BAD ARGUMENT (11) for zero step; CONTROL STACK FULL (18); LOOP VARIABLE IN USE (19); PROGRAM ONLY (16).

GOSUB

Syntax: `GOSUB line`

Parameters: Literal existing line 1–65535; program statement.

Result: statement; no value.

Example fragment: `10 GOSUB 100`

Saves current source location for RETURN, including a colon continuation. Shares eight control frames.

Errors: UNDEFINED LINE (12); CONTROL STACK FULL (18); PROGRAM ONLY (16); BAD LINE NUMBER (3).

GOTO

Syntax: `GOTO line`

Parameters: Literal existing line 1–65535; program statement.

Result: statement; no value.

Example fragment: `20 GOTO 10`

Jumps without saving a return location. A variable/expression is not a line-target substitute.

Errors: UNDEFINED LINE (12); PROGRAM ONLY (16); BAD LINE NUMBER (3).

HELP

Syntax: `HELP [page]`

Parameters: Direct command; literal page 1–9; omitted means 1.

Result: statement; no value.

Example fragment: `HELP 8`

Native text reference: strings/input 7, input codes 8, functions/editing 9. It does not open this host PDF.

Errors: BAD ARGUMENT (11); DIRECT COMMAND ONLY (6); SYNTAX ERROR (1).

HIDE

Syntax: `HIDE slot`

Parameters: Numeric sprite slot 0–63.

Result: statement; no value.

Example fragment: `HIDE 0`

Safe for an already-hidden slot; permitted in either text or graphics mode.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

IF

Syntax: `IF condition THEN line-or-statement`

Parameters: Numeric condition; string comparisons produce numeric truth. Program statement.

Result: statement; no value.

Example fragment: `10 IF N$="" THEN PRINT "EMPTY"`

Nonzero is true. A false condition skips the rest of its whole line. Inline THEN and literal targets supported; no ELSE.

Errors: PROGRAM ONLY (16); UNDEFINED LINE (12); TYPE MISMATCH (25); syntax/range errors in expressions.

INKEY$

Syntax: `INKEY$`

Parameters: No arguments or parentheses.

Result: string.

Example fragment: `K$=INKEY$`

Nonblocking one queued event byte or empty string. In RUN, reserved button edge codes 128–135. Eight-event ring; drop newest on overflow.

Errors: STRING SPACE FULL (26) on temporary exhaustion. Empty is normal; ASC(empty) is not.

INPUT

Syntax: `INPUT [prompt;]A or INPUT [prompt;]A$`

Parameters: Optional string prompt; one numeric/string variable.

Result: statement; no value.

Example fragment: `INPUT "NAME";N$`

Uses 95-character line editor. Empty strings valid; numeric empty/invalid retries. String limit 63/shared pool applies.

Errors: BAD INPUT (17); STRING TOO LONG (27); STRING SPACE FULL (26); BREAK (8); TYPE MISMATCH (25).

INT

Syntax: `INT(N)`

Parameters: N: integer.

Result: integer.

Example fragment: `PRINT INT(-7)`

Identity in an integer-only language; does not enable fractional literals or floating point.

Errors: TYPE MISMATCH (25); expression syntax/range errors.

JOY

Syntax: `JOY(port)`

Parameters: Numeric port 1 or 2.

Result: integer.

Example fragment: `J=JOY(1)`

Held-button sum:1/2/4/8/16/32/64/128. Port 2 returns 0 after optional external-keyboard handshake, until reset.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

KILL

Syntax: `KILL ["NAME" | slot]`

Parameters: Direct command; name rules match SAVE, slot 1–6.

Result: statement; no value.

Example fragment: `KILL 1`

Confirmed deletion, alias of ERASE. Numbered slot can explicitly delete a damaged entry. No automatic reclamation.

Errors: FILE NOT FOUND (22); BAD ARGUMENT (11); DIRECT COMMAND ONLY (6).

LEFT$

Syntax: `LEFT$(S$,N)`

Parameters: String and nonnegative numeric count.

Result: string.

Example fragment: `A$=LEFT$("HELLO",2)`

Returns up to N bytes from the start. Zero is empty; oversized count clamps to string length.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25); destination/temporary string limits 26/27.

LEN

Syntax: `LEN(S$)`

Parameters: String expression.

Result: integer.

Example fragment: `A=LEN("HELLO")`

Counts bytes, including embedded NUL. Stored variables max 63, but a literal can use the line limit.

Errors: TYPE MISMATCH (25); expression errors.

LET

Syntax: `LET variable=expression`

Parameters: A–Z numeric or A$–Z$ string variable with matching typed expression.

Result: statement; no value.

Example fragment: `LET A$="HELLO"`

LET is optional. Independent numeric/string variables; no long identifiers, arrays or implicit conversion.

Errors: TYPE MISMATCH (25); STRING SPACE FULL (26); STRING TOO LONG (27); expression errors.

LINE

Syntax: `LINE x1,y1,x2,y2[,ink]`

Parameters: Signed numeric endpoints, optional ink 0/1 default 1.

Result: statement; no value.

Example fragment: `LINE 0,0,127,63`

SCREEN 1; inclusive integer Bresenham line, each point clips to 128×64 canvas. Very long segments can take many frames.

Errors: GRAPHICS MODE REQUIRED (20); BAD ARGUMENT (11); TYPE MISMATCH (25).

LIST

Syntax: `LIST`

Parameters: No arguments; direct command.

Result: statement; no value.

Example fragment: `LIST`

Displays tokenized source in numeric order. Long output pages: Start continues, Select cancels.

Errors: DIRECT COMMAND ONLY (6); BREAK (8); BAD PROGRAM (7) if source is invalid.

LOAD

Syntax: `LOAD ["NAME"]`

Parameters: Direct command; quoted 1–12-character name or default PROGRAM.

Result: statement; no value.

Example fragment: `LOAD "HELLO"`

Validates source/art before replacement; confirms dirty working data. Clears values, restores saved art. Read versions 2/3.

Errors: FILE NOT FOUND (22); BAD SAVED PROJECT (23); DIRECT COMMAND ONLY (6); BREAK (8).

LOADDEMO

Syntax: `LOADDEMO number`

Parameters: Literal built-in demo 1–6; direct command.

Result: statement; no value.

Example fragment: `LOADDEMO 6`

Replaces working source after confirmation if dirty. Retains kept artwork, uses no save slot. Demo 6 is input diagnostic.

Errors: BAD ARGUMENT (11); DIRECT COMMAND ONLY (6); BREAK (8).

LOCATE

Syntax: `LOCATE column,row`

Parameters: Numeric column 0–27, row 0–21.

Result: statement; no value.

Example fragment: `LOCATE 4,3`

Sets text output position. Restores text if currently in graphics; graphics surface is cleared in that transition.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

MID$

Syntax: `MID$(S$,start[,count])`

Parameters: String, numeric positive one-based start, optional nonnegative count.

Result: string.

Example fragment: `A$=MID$("HELLO",2,3)`

Count defaults to rest. Past end returns empty; count clamps to available bytes.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25); string destination/temporary limits 26/27.

MOVE

Syntax: `MOVE slot,x,y`

Parameters: Numeric visible slot 0–63, x 0–255, y 1–239.

Result: statement; no value.

Example fragment: `MOVE 0,88,100`

SCREEN 1 required; preserves slot artwork. A hidden slot is an invalid MOVE target.

Errors: BAD ARGUMENT (11); GRAPHICS MODE REQUIRED (20); TYPE MISMATCH (25).

NEW

Syntax: `NEW`

Parameters: No arguments; direct command.

Result: statement; no value.

Example fragment: `NEW`

Confirms nonempty dirty source; resets program/values, stops sound/hides sprites. Keeps kept artwork. Does not erase saved files.

Errors: DIRECT COMMAND ONLY (6); SYNTAX ERROR (1).

NEXT

Syntax: `NEXT A`

Parameters: Numeric variable used by the active FOR.

Result: statement; no value.

Example fragment: `40 NEXT I`

Advances by STEP and loops or finishes. Active frame must match; variable remains at its post-loop value.

Errors: UNMATCHED LOOP (14); OVERFLOW (9); PROGRAM ONLY (16); TYPE MISMATCH (25).

PRINT

Syntax: `PRINT [expression[;|,expression]...]`

Parameters: Numeric/string expressions; optional trailing separator.

Result: statement; no value.

Example fragment: `PRINT "VALUE=";A`

Semicolon joins, comma uses four-column tab stops. Default ends line; trailing separator preserves position. Restores text from graphics.

Errors: OUTPUT TOO LONG (5); TYPE MISMATCH (25); expression errors; BREAK (8) on canceled pages.

PSET

Syntax: `PSET x,y[,ink]`

Parameters: Signed numeric local coordinates, optional ink 0/1 default 1.

Result: statement; no value.

Example fragment: `PSET 64,32`

SCREEN 1; set/clear a point in 128×64 canvas. Out-of-canvas coordinates clip to no-op.

Errors: GRAPHICS MODE REQUIRED (20); BAD ARGUMENT (11); TYPE MISMATCH (25).

RECT

Syntax: `RECT x1,y1,x2,y2[,ink]`

Parameters: Numeric opposite corners, optional ink 0/1 default 1.

Result: statement; no value.

Example fragment: `RECT 8,8,119,55`

SCREEN 1; outline, not filled rectangle or x/y/width/height. Reversed and degenerate corners supported; clipping applies.

Errors: GRAPHICS MODE REQUIRED (20); BAD ARGUMENT (11); TYPE MISMATCH (25).

REM

Syntax: `REM comment`

Parameters: Remaining characters on the source line.

Result: statement; no value.

Example fragment: `10 REM A IS THE RUNNING TOTAL`

Comment consumes the rest of the line, including colon/keywords. It is not an executable statement separator.

Errors: Physical/source memory bounds still apply.

RETURN

Syntax: `RETURN`

Parameters: No arguments; matching GOSUB frame.

Result: statement; no value.

Example fragment: `110 RETURN`

Resumes the saved source/colon continuation. Natural source EOF with pending GOSUB also reports an invalid return.

Errors: INVALID RETURN (13); PROGRAM ONLY (16).

RIGHT$

Syntax: `RIGHT$(S$,N)`

Parameters: String and nonnegative numeric count.

Result: string.

Example fragment: `A$=RIGHT$("HELLO",2)`

Returns up to N bytes from end. Zero returns empty; oversized count clamps to available bytes.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25); string destination/temporary limits 26/27.

RND

Syntax: `RND(N)`

Parameters: Numeric N 1–32767.

Result: integer.

Example fragment: `A=RND(6)+1`

Returns 0 through N-1 using deterministic 16-bit LFSR. Variable reset seeds ACE 1; RUN repeats initial sequence. No RANDOMIZE.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

RUN

Syntax: `RUN ["NAME"]`

Parameters: Whole direct command; optional quoted saved name.

Result: statement; no value.

Example fragment: `RUN "HELLO"`

Starts from beginning with reset values. Named form validates/loads source/art and confirms dirty replacement. Does not SAVE.

Errors: FILE NOT FOUND (22); BAD SAVED PROJECT (23); DIRECT COMMAND ONLY (6); runtime errors from program.

SAVE

Syntax: `SAVE ["NAME"]`

Parameters: Whole direct command; optional quoted name, default PROGRAM.

Result: statement; no value.

Example fragment: `SAVE "HELLO"`

Six slots, names 1–12 uppercase letters/digits/_/./-. Source and kept art only. Confirm overwrite; wait for SAVED notice. Writes version 3.

Errors: SAVE DIRECTORY FULL (24); BAD PROGRAM (7); DIRECT COMMAND ONLY (6); BREAK (8).

SCREEN

Syntax: `SCREEN mode`

Parameters: Numeric 0=text,1=graphics.

Result: statement; no value.

Example fragment: `SCREEN 1`

Graphics start clears canvas/restores tile mapping/default colors. Return to text clears canvas. Hides sprites during screen transition.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

SGN

Syntax: `SGN(N)`

Parameters: Numeric expression.

Result: integer.

Example fragment: `A=SGN(-42)`

Returns -1 for negative,0 for zero,1 for positive. No floating-point conversion.

Errors: TYPE MISMATCH (25); expression errors.

SOUND

Syntax: `SOUND channel,pitch,duration`

Parameters: Numeric channel 1–4; tonal notes 33–96 for 1–3, noise index 0–15 for 4; duration 0–32767 frames.

Result: statement; no value.

Example fragment: `SOUND 1,69,60`

Asynchronous, replaces selected voice; duration 0 mutes it. Fixed levels; no DMC/samples. NTSC pitch target.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

SPRCOL

Syntax: `SPRCOL(a,b)`

Parameters: Numeric sprite slots 0–63.

Result: integer.

Example fragment: `A=SPRCOL(0,1)`

1 when both visible unclipped 8×8 boxes overlap,0 otherwise. Touching edges do not overlap; transparent pixels still count.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25).

SPRITE

Syntax: `SPRITE slot,x,y,art`

Parameters: Numeric slot 0–63, x 0–255, y 1–239, art 0–39.

Result: statement; no value.

Example fragment: `SPRITE 0,80,100,0`

SCREEN 1 required. Native 8×8 hardware sprite; art 0–7 predefined,8–39 kept editable. Eight per scanline.

Errors: BAD ARGUMENT (11); GRAPHICS MODE REQUIRED (20); TYPE MISMATCH (25).

STEP

Syntax: `FOR A=start TO limit STEP step`

Parameters: Numeric nonzero step in FOR syntax.

Result: grammar fragment.

Example fragment: `10 FOR I=5 TO 1 STEP -1`

FOR grammar fragment, not a standalone command. Default step is 1; negative steps supported.

Errors: BAD ARGUMENT (11) for zero step; FOR/control errors.

STOP

Syntax: `STOP`

Parameters: No arguments.

Result: statement; no value.

Example fragment: `50 STOP`

Intentional program termination with control-frame cleanup and sound stop, like END; no CONT command is supplied.

Errors: SYNTAX ERROR (1).

STR$

Syntax: `STR$(N)`

Parameters: Signed integer expression.

Result: string.

Example fragment: `A$=STR$(-42)`

Decimal text, minus sign for negatives, no positive leading space. Does not enable decimal fractions.

Errors: TYPE MISMATCH (25); STRING SPACE FULL (26); expression errors.

THEN

Syntax: `IF condition THEN line-or-statement`

Parameters: Part of IF grammar.

Result: grammar fragment.

Example fragment: `10 IF A=1 THEN PRINT "YES"`

Not a standalone command. Literal target or inline statement; false skips whole remainder of line. No ELSE.

Errors: IF/program/target/expression errors.

TILE

Syntax: `TILE column,row,id`

Parameters: Numeric column 0–31, row 0–29, pattern 0–255.

Result: statement; no value.

Example fragment: `TILE 0,0,65`

SCREEN 1 native nametable placement, no palette argument. Patterns share; drawing changes referenced canvas patterns.

Errors: BAD ARGUMENT (11); GRAPHICS MODE REQUIRED (20); TYPE MISMATCH (25).

TO

Syntax: `FOR A=start TO limit [STEP step]`

Parameters: Numeric inclusive limit within FOR.

Result: grammar fragment.

Example fragment: `10 FOR I=1 TO 5`

FOR grammar fragment, not a standalone command. Initial value past limit can yield zero trips.

Errors: FOR/program/expression errors.

VAL

Syntax: `VAL(S$)`

Parameters: String containing whole signed decimal with optional surrounding spaces.

Result: integer.

Example fragment: `A=VAL(" -42 ")`

Strict parse, not prefix conversion. Empty, trailing garbage and embedded NUL invalid; range -32768…32767.

Errors: BAD ARGUMENT (11); OVERFLOW (9); TYPE MISMATCH (25).

WAIT

Syntax: `WAIT frames`

Parameters: Numeric 0–32767 emulated frames.

Result: statement; no value.

Example fragment: `WAIT 60`

Yields interruptibly; zero adds no delay. Counts emulated time, freezes on app pause. Other statements add their own time.

Errors: BAD ARGUMENT (11); TYPE MISMATCH (25); BREAK (8).

15. Errors, recovery and troubleshooting

Read the exact message first. During RUN the suffix IN line identifies the source line. Return to editing, correct that numbered line, and RUN again. Failed line storage keeps the previous valid record. Failed typed assignment keeps its old destination. Reset is not the first repair step: it discards unsaved work.

SYNTAX ERROR often means missing quotes, a parenthesis/comma, extra punctuation, a wrong keyword or an unsupported identifier. BAD NUMBER is a literal outside the numeric domain; OVERFLOW is arithmetic/conversion beyond the checked range. TYPE MISMATCH needs an explicit STR$/VAL conversion or a corrected variable type. BAD INPUT retries an edited numeric answer. A bad string INPUT can retry after STRING TOO LONG or STRING SPACE FULL. Empty strings are valid input but not valid arguments to ASC or VAL.

UNDEFINED LINE means a GOTO/GOSUB/THEN target does not exist. INVALID RETURN means there is no matching GOSUB, or source ended while one was pending. UNMATCHED LOOP means FOR/NEXT frames do not match, including missing NEXT. LOOP VARIABLE IN USE means a nested FOR reuses an active numeric loop variable. CONTROL STACK FULL or EXPRESSION TOO DEEP calls for simpler nesting, not a larger host heap.

GRAPHICS MODE REQUIRED needs SCREEN 1 before drawing/SPRITE/MOVE. BAD ARGUMENT usually signals a range, count, mode parameter or invalid hidden-slot MOVE. FILL TOO COMPLEX is the bounded seed worklist, not a host crash. FILE NOT FOUND, BAD SAVED PROJECT and SAVE DIRECTORY FULL need the file checks in Chapter 12. BREAK is the expected result of an interruption or canceled output page.

Code	Exact message
1	SYNTAX ERROR
2	OUT OF MEMORY
3	BAD LINE NUMBER
4	BAD NUMBER
5	OUTPUT TOO LONG
6	DIRECT COMMAND ONLY
7	BAD PROGRAM
8	BREAK
9	OVERFLOW
10	DIVISION BY ZERO
11	BAD ARGUMENT
12	UNDEFINED LINE

Code	Exact message
13	INVALID RETURN
14	UNMATCHED LOOP
15	EXPRESSION TOO DEEP
16	PROGRAM ONLY
17	BAD INPUT
18	CONTROL STACK FULL
19	LOOP VARIABLE IN USE
20	GRAPHICS MODE REQUIRED
21	FILL TOO COMPLEX
22	FILE NOT FOUND
23	BAD SAVED PROJECT
24	SAVE DIRECTORY FULL
25	TYPE MISMATCH
26	STRING SPACE FULL
27	STRING TOO LONG

If a controller seems stuck, release it, check whether the on-screen keyboard has focus, and use the app PAUSE control. A disconnected gamepad pauses the app. Text typing belongs to BASIC only while that ROM is running. Opening

this host guide pauses and isolates input; close it and explicitly Resume to continue. A PDF view failure does not imply that the NES ROM is corrupt; use Help's safe Back/Close action and its local export option if available.

16. Appendices: codes, limits and edition history

Printable text follows ASCII where the font/editor support it: space 32, quotation mark 34, digits 48–57, uppercase letters 65–90. Keyboard source entry normalizes lowercase to uppercase. CHR$ can create all bytes 0–255, but unsupported or control glyphs are not promised visible. LEN counts bytes, not Unicode characters.

The relevant control bytes are Backspace 8, Enter 13, Escape 27 and the optional bridge's caret controls 28–31. Ctrl+C is byte 3 and always interrupts rather than becoming an ordinary queued text event. The reserved controller-event table is:

Button	INKEY$ code	JOY bit
A	128	1
B	129	2
Select	130	4
Start	131	8
Up	132	16
Down	133	32
Left	134	64
Right	135	128

Resource	Limit
Numeric variables	26, A–Z, signed 16-bit
String variables	26, A$–Z$, independent
Stored/concatenated string	63 bytes
Shared string pool	256 bytes
String expression arena	128 bytes
Physical input line	95 characters
Program	128 lines / 3072 tokenized bytes
Control frames	8 shared FOR/GOSUB frames
Canvas	128×64, one bit per pixel
Sprite slots	64, eight per scanline
Editable art	32 patterns, IDs 8–39, 8×8
Project files	Six named slots

Supported: integer math, typed strings, the reference's functions, native input and controller events, text I/O, line/branch/loop/subroutine control, one-bit canvas, hardware sprites/editor, asynchronous APU sound and explicit project storage. Unsupported: long variable names, arrays, floats, ELSE, DATA/READ, ON GOTO, arbitrary gamepad scan codes,

Unicode source, arbitrary sprite palettes, composed-sprite editing or inter-ROM project transfer. Future features are not promised by this edition.

Edition 1.0 is the first bundled host programming book, dated 3 November 2026. It covers the completed BASIC language in app 0.7.1(30), with unchanged ROM bytes from completion build 0.7.0(29). The exact fingerprint appears at the front of this book and in the app's local guide metadata. The standalone ROM uses NES 2.0, mapper 1/submapper 0, 128 KiB PRG ROM, 8 KiB CHR RAM and 32 KiB PRG NVRAM, NTSC.

Physical NES/flashcart acceptance remains NOT VERIFIED. Windows/macOS PDF readers can open an exported copy of the book; that does not imply a separate native 8-Bit Impostor app exists on those platforms. The app's actual supported build is iOS/iPadOS 17 or later. iOS-app compatibility on Mac is not certified here.

See ABOUT / SOFTWARE NOTICES in the app for project/core/font licenses. No public product-support URL or email was confirmed for this edition; none is invented in this book. Original NDB software is independent of Nintendo. Use only ROMs you have rights to use. The guide's pixel artwork is original, and the embedded reading/code fonts retain their redistribution licenses with the build source.