

OSAWARE

BASIC Language Reference Manual

```
10 GL.INIT : GL.PERSPECTIVE 55  
20 GL.SPHERE 1.0, 32, 24  
30 S=GL.MESHID : GL.CHROME S, 0.05  
40 ON MOUSE GOSUB 500 : MOUSE ON  
50 GL.FOG 5,5,18,6,22 : GL.DRAWALL
```

The Geekprocessor v0.5 — Build Q90

Copyright © 2026 Exedos — All rights reserved

Contents

| | | |
|-----|---|----|
| 1. | Getting Started — Tutorials | 3 |
| 2. | Language Fundamentals | 6 |
| 3. | Core Commands | 7 |
| 4. | Flow Control | 8 |
| 5. | String Functions | 9 |
| 6. | Math Functions | 10 |
| 7. | Arrays, DATA and READ | 11 |
| 8. | 2D Graphics | 12 |
| 9. | Images | 13 |
| 10. | Mouse Input | 14 |
| 11. | Sprites — OBJECT System | 15 |
| 12. | 3D Graphics — GL System | 16 |
| 13. | Audio — SOUND & WAVE ★ New | 19 |
| 14. | Subprograms — SUB & FUNCTION ★ New | 20 |
| 15. | Networking — WebSocket & SSH | 22 |
| 16. | AI Commands | 23 |
| 17. | System Commands | 24 |
| 18. | Colour Palette | 25 |
| 19. | Quick Reference Card | 26 |

1. Getting Started — Tutorials

These short programs introduce key OSAWARE BASIC concepts.

Tutorial 1 — Hello, World!

```
10 REM My first program
20 CLS
30 COLOUR 3
40 PRINT "Hello, world!"
50 PRINT "Welcome to OSAWARE BASIC"
60 END
```

- Type each line with its number, press Enter after each, then type RUN.

Tutorial 2 — Variables and Input

```
10 CLS : COLOUR 3
20 INPUT "What is your name? "; NAME$
30 INPUT "How old are you? "; AGE
40 PRINT "Hello, "; NAME$; "!"
50 NEXT_YEAR = AGE + 1
60 PRINT "Next year you will be "; NEXT_YEAR
70 END
```

Tutorial 3 — Loops and Counting

```
10 CLS : COLOUR 3
20 PRINT "Multiplication table for 7:"
30 FOR I = 1 TO 12
40   PRINT "7 x "; I; " = "; 7*I
50 NEXT I
60 END
```

Tutorial 4 — Subroutines

```
10 CLS : COLOUR 3
20 NUM = 8 : GOSUB 1000
30 PRINT "Factorial of "; NUM; " = "; RESULT
40 END
1000 RESULT = 1
1010 FOR I = 1 TO NUM
1020   RESULT = RESULT * I
1030 NEXT I
1040 RETURN
```

Tutorial 5 — Simple Graphics

```
10 CLS : DELAY 0
```

```

20 FOR I = 1 TO 60
30   X = 100 + I * 4
40   Y = 150 + SIN(I * 0.2) * 80
50   CIRCLE X, Y, 8, (I MOD 15) + 1
60 NEXT I
70 END

```

■ Use `DELAY 0` for fast batch graphics. Use `SLEEP` to slow things down for animation.

Tutorial 6 — Game Loop with Mouse

```

10 CLS : COLOUR 3
20 PRINT "Click to draw! Q to quit."
30 ON MOUSE GOSUB 1000 : MOUSE ON
40 K = INKEY : IF K<>81 AND K<>113 THEN SLEEP 30 : GOTO 40
50 MOUSE OFF : END
1000 B = MOUSE(0)
1010 IF B=-1 OR B=1 THEN CIRCLE MOUSE(3),MOUSE(4),8,INT(RND(1)*14)+1
1020 RETURN

```

Tutorial 7 — 3D Graphics with GL

```

10 GL.INIT : GL.PERSPECTIVE 55
20 GL.CAMERA 0, 0, -5 : GL.LOOKAT 0, 0, 0
30 GL.SOLIDWIRE : GL.COLOUR 80, 140, 255
40 GL.LIGHT 1, 2, -1 : GL.AMBIENT 0.2 : GL.SHINE 60
50 GL.BEGIN
60   GL.VERTEX -1,-1,-1 : GL.VERTEX 1,-1,-1
70   GL.VERTEX 1, 1,-1 : GL.VERTEX -1, 1,-1
80   GL.VERTEX -1,-1, 1 : GL.VERTEX 1,-1, 1
90   GL.VERTEX 1, 1, 1 : GL.VERTEX -1, 1, 1
100  GL.FACE 1,2,3,4 : GL.FACE 8,7,6,5
110  GL.FACE 1,4,8,5 : GL.FACE 2,3,7,6
120  GL.FACE 4,3,7,8 : GL.FACE 1,2,6,5
130 GL.END : C = GL.MESHID
140 A = 0
150 K = INKEY : IF K=81 THEN END
160 GL.CLS 5,5,20 : GL.ROTATE C,A*0.7,A,0 : GL.DRAWALL
170 A = A+1.5 : SLEEP 16 : GOTO 150

```

■ Try: `RUN GLDEMO` `RUN GL3D` `RUN GLSHADE` `RUN SGIDEMO` `RUN GLDEMO3`

Tutorial 8 — Chrome Sphere with Fog

```

10 GL.INIT : GL.PERSPECTIVE 55
20 GL.CAMERA 0, 2, -8 : GL.LOOKAT 0, 0, 0
30 GL.SOLID : GL.LIGHT 1,2,-1 : GL.AMBIENT 0.2
40 GL.FOG 5, 5, 18, 4, 16

```

```
50 GL.POINTLIGHT 0, 4, 0, 255, 200, 100, 4, 12
60 GL.COLOUR 210, 225, 255 : GL.SHINE 200
70 GL.SPHERE 1.2, 48, 32
80 S = GL.MESHID : GL.CHROME S, 0.05
90 GL.SOLID : GL.COLOUR 255,255,255 : GL.SHINE 5
100 GL.BEGIN
110 GL.VERTEX -1,0,-1 : GL.VERTEX 1,0,-1
120 GL.VERTEX 1,0,1 : GL.VERTEX -1,0,1
130 GL.FACE 1,2,3,4 : GL.END
140 F = GL.MESHID : GL.TRANSLATE F,0,-1.5,0
150 GL.SCALE F,10,1,10 : GL.TEXTURE F,"checkerboard",6
160 A=0
170 K=INKEY : IF K=81 THEN END
180 GL.CLS 5,5,18 : SY = SIN(A*0.05)*0.6+0.1
190 GL.TRANSLATE S, 0, SY, 0 : GL.DRAWALL
200 A=A+1 : SLEEP 16 : GOTO 170
```

2. Language Fundamentals

Program Structure

Every line begins with a line number (0–9999). Lines execute in ascending order. Multiple statements on one line are separated by colons.

```
10 REM Single statement
20 X=1 : Y=2 : Z=3
```

| Type | Name examples | Notes |
|-----------|------------------|-------------------------------|
| Numeric | A, NR, MAXI, ZR2 | Default 0. Floating point. |
| String | A\$, NAME\$, C\$ | Default "". Must end with \$. |
| Array | DIM A(10) | Numeric array, indices 0-10. |
| Str Array | DIM N\$(5) | String array. |

Operators

Arithmetic: + - * / ^ % (power, modulo)

Comparison: = <> < > <= >=

Logical: AND OR NOT

String: + (concatenation)

■ *Variable names are case-insensitive. A and a refer to the same variable.*

3. Core Commands

PRINT — Output text or values to the terminal

| | |
|----------------|---|
| Syntax | PRINT [expr] [; expr] [, expr] |
| Example | <pre>PRINT "Hello, World!" PRINT "X="; X; " Y="; Y PRINT TAB\$(10); "Indented" PRINT</pre> |
| Idiom | <i>Semicolons suppress spacing. A trailing semicolon suppresses the newline. PRINT alone prints a blank line.</i> |

INPUT — Accept input from the user

| | |
|----------------|--|
| Syntax | INPUT ["prompt";] var [, var2] |
| Example | <pre>INPUT X INPUT "Enter name: "; NAME\$ INPUT "W, H: "; W, H</pre> |
| Idiom | <i>Always provide a prompt string. INPUT halts until the user presses Enter.</i> |

LET — Assign a value (LET is optional)

| | |
|----------------|--|
| Syntax | LET var = expr |
| Example | <pre>X = 42 ' LET optional A\$ = "hello" : X = X + 1 ' colon separates</pre> |

DATA / READ / RESTORE — Literal data in program

| | |
|----------------|--|
| Syntax | <pre>DATA val, val, "str" READ var, var2 RESTORE [line]</pre> |
| Example | <pre>100 DATA 10, 20, "Alice" 110 READ X, Y, N\$ 120 RESTORE</pre> |

| Command / Syntax | Description |
|------------------|-------------------------------------|
| END | Stop program |
| REM text | Comment |
| CLS | Clear terminal and canvas |
| COLOUR n | Set text/graphics colour (1-16) |
| SWAP A, B | Swap two variables |
| MEM / INFO | Program info: line count, variables |

4. Flow Control

| Command / Syntax | Description |
|--|-------------------------|
| GOTO n | Jump to line n or label |
| IF cond THEN stmt | Single-line conditional |
| IF cond THEN\n ... \nEND IF | Block IF |
| IF c THEN s ELSE t | With else branch |
| GOSUB n / RETURN | Call subroutine at n |
| FOR I=a TO b STEP s\ n ... \nNEXT I | Count loop |
| WHILE cond\n ... \nWEND | Condition loop |
| ON n GOTO l1,l2 | Jump to line ln |
| ON n GOSUB l1,l2 | Call subroutine ln |
| ON ERROR GOTO n | Error handler |

Examples

```
IF X=42 THEN PRINT "Found!"
IF A$="yes" THEN GOTO 500
FOR I = 10 TO 1 STEP -1 : PRINT I : NEXT I
WHILE ZR*ZR+ZI*ZI<4 : TMP=ZR*ZR-ZI*ZI+CR : ZI=2*ZR*ZI+CI : ZR=TMP : WEND
```

■ *Avoid IF THEN GOTO inside tight loops — use WHILE or FOR instead.*

5. String Functions

| Command / Syntax | Description |
|-----------------------------|-------------------------------------|
| LEN(s\$) | Length of string |
| LEFT\$(s\$,n) | First n characters |
| RIGHT\$(s\$,n) | Last n characters |
| MID\$(s\$,start[,len]) | 1-based substring |
| UPPER\$(s\$) / UCASE\$(s\$) | Upper case |
| LOWER\$(s\$) | Lower case |
| STR\$(n) | Number to string |
| VAL(s\$) | String to number |
| CHR\$(n) | Character from ASCII code |
| ASC(s\$) | ASCII code of first char |
| INSTR(s\$,find\$) | Position of find\$ (0 if not found) |
| SPACE\$(n) | n spaces |
| STRING\$(n,c\$) | n copies of c\$ |
| TAB\$(n) | n spaces for PRINT alignment |
| HEX\$(n) | Hex string |
| CENTER\$(s\$[,w]) | Centre-pad to width |

```
MID$("Hello World", 7, 5) ' "World"
INSTR("Hello", "ll")    ' 3
VAL(" 42abc")           ' 42
```

6. Math Functions

| Command / Syntax | Description |
|----------------------|--|
| INT(n) | Floor toward $-\infty$ |
| ABS(n) | Absolute value |
| SGN(n) | -1, 0, or 1 |
| SQR(n) | Square root |
| LOG(n) | Natural log |
| EXP(n) | e^n |
| SIN(r)/COS(r)/TAN(r) | Trig (radians) |
| ATN(n) | Arc tangent |
| RND(n) | Float 0..1 for $n \leq 1$, integer 0..n-1 for $n > 1$ |
| RANDOMIZE [TIMER] | Seed RNG |

```
PI = 4 * ATN(1)
D6 = INT(RND(1)*6) + 1 'dice roll
```

7. Arrays, DATA and READ

Arrays

```
DIM A(10)
A(0) = 100 : A(I) = A(I)+1
PRINT UBOUND(A) ' = 10
```

■ *DIM A(10) gives valid indices 0 through 10.*

DATA / READ

```
100 DATA 10, 20, "Alice"
110 READ X, Y, N$
120 RESTORE
```

8. 2D Graphics

All 2D commands draw on the canvas behind the terminal. The first graphics command activates graphics mode. CLS clears back to black.

| Command / Syntax | Description |
|--------------------------------|---------------------------------------|
| PSET x,y [,col] | Draw a pixel |
| PRESET x,y | Erase a pixel |
| LINE x1,y1,x2,y2 [,col] | Draw a line |
| CIRCLE x,y,r [,col] | Draw a circle outline |
| RECT x1,y1,x2,y2 [,col] | Rectangle outline |
| FILLRECT x1,y1,x2,y2 [,col] | Solid filled rectangle |
| PAINT x,y [,col] | Flood fill |
| COLOUR n [,bg] | Set foreground/background colour |
| CLS | Clear screen |
| DELAY ms | Per-statement delay (0=batch/fastest) |
| SLEEP ms | Pause once for ms milliseconds |
| WIDTH / HEIGHT | Canvas size in pixels (read-only) |
| POINT(x,y) | Return colour index of pixel |

```
CIRCLE 200,150,50,3 : PAINT 200,150,3 'filled circle
FILLRECT 10,10,200,100,8 'solid rectangle
```

9. Images

| Command / Syntax | Description |
|--------------------------------|---|
| LOADIMG "name", "url" | Fetch and cache image (pauses until loaded) |
| DISPLAY "name" [,x,y[,w,h]] | Draw stored image to canvas |
| IMGLIST | List all stored images |
| IMGFREE "name" | Remove from store |
| IMAGE x,y,url\$ [,w,h] | Direct draw from URL |
| OBJECT.SHAPE id,"name" | Use image as sprite shape |

Built-in images (always available offline)

| Name | Size | Description |
|--------------|-------|------------------------------------|
| checkerboard | 32×32 | Grey checker pattern |
| gradient | 64×32 | Rainbow spectrum |
| smiley | 32×32 | RGBA yellow smiley, transparent bg |
| palette | 64×64 | 16-colour block grid |

```
DISPLAY "gradient", 0, 0
DISPLAY "smiley", 100, 100, 64, 64
```

10. Mouse Input

OSAWARE BASIC provides Amiga BASIC compatible mouse input. Events fire a subroutine via ON MOUSE GOSUB when the left button is pressed.

Commands

| Command / Syntax | Description |
|--------------------|--|
| MOUSE ON | Enable mouse event trapping |
| MOUSE OFF | Disable event trapping |
| MOUSE STOP | Suspend (queue events, don't fire GOSUB) |
| ON MOUSE GOSUB lbl | Call subroutine on left button press |

MOUSE(n) function

| Command / Syntax | Description |
|------------------|---|
| MOUSE(0) | Button status: 0=up 1=click 2=dblclick -1=held. Clears on read (except -1). |
| MOUSE(1) | Current cursor X (canvas-relative pixels) |
| MOUSE(2) | Current cursor Y |
| MOUSE(3) | X where button was last pressed |
| MOUSE(4) | Y where button was last pressed |
| MOUSE(5) | X at last button release |
| MOUSE(6) | Y at last button release |

```

10 ON MOUSE GOSUB 1000 : MOUSE ON
20 K=INKEY : IF K<>81 THEN SLEEP 30 : GOTO 20
30 MOUSE OFF : END
1000 B=MOUSE(0)
1010 IF B=-1 OR B=1 THEN CIRCLE MOUSE(3),MOUSE(4),8,3
1020 RETURN

```

■ *MOUSE(0) returns -1 while held — use for drag detection. Check MOUSE(3)/MOUSE(4) for click position.*

11. Sprites — OBJECT System

The OBJECT system provides Amiga BASIC-compatible hardware sprites. Each object has a pixel shape, position, velocity, and priority. Transparent pixels use colour code 000000.

Shape Definition

```
REM Format: "width,height,RRGGBBRRGGBB..."
REM 000000 = transparent pixel
OBJECT.SHAPE 1, "16,16,FFDD00FFDD00...000000..."
LOADIMG "player", "https://host/sprite.png"
OBJECT.SHAPE 1, "player"
```

| Command / Syntax | Description |
|--------------------------------|---|
| OBJECT.SHAPE id, def\$ | Set sprite shape (hex string or image name) |
| OBJECT.ON [id] | Show sprite |
| OBJECT.OFF [id] | Hide sprite |
| OBJECT.X id, val | Set X position |
| OBJECT.Y id, val | Set Y position |
| OBJECT.VX id, val | X velocity (px/sec) |
| OBJECT.VY id, val | Y velocity |
| OBJECT.AX/AY id, val | X/Y acceleration |
| OBJECT.PRIORITY id, n | Z-order (higher = drawn on top) |
| OBJECT.START | Start 60fps animation timer |
| OBJECT.STOP | Stop timer |
| OBJECT.CLOSE [id] | Remove sprite(s) |
| COLLISION ON/OFF/STOP | Enable/disable collision detection |
| ON COLLISION GOSUB lbl | Fire subroutine on collision |
| OBJECT.X(id) / OBJECT.Y(id) | Read current position |
| COLLISION(id) | ID of object colliding with id |

■ For game-loop movement, use OBJECT.X/Y directly without OBJECT.START. OBJECT.X erases at the old position and redraws at the new one automatically.

12. 3D Graphics — GL System (Three.js WebGL)

The GL system renders 3D scenes using Three.js WebGL — GPU-accelerated with Phong shading, real-time reflections, depth fog, and point lights.

Scene Setup

| Command / Syntax | Description |
|--------------------|---|
| GL.INIT | Create WebGL renderer, scene, camera and lights |
| GL.CLS [r,g,b] | Clear canvas (0-255 each, default black) |
| GL.PERSPECTIVE fov | Field of view in degrees (e.g. 55) |
| GL.CAMERA x,y,z | Camera world position |
| GL.LOOKAT x,y,z | Point camera looks at (default 0,0,0) |

Render Modes

| Command / Syntax | Description |
|------------------|-----------------------------|
| GL.WIRE | Wireframe edges only |
| GL.SOLID | Solid Phong shading |
| GL.SOLIDWIRE | Solid Phong + edge outlines |

Lighting

| Command / Syntax | Description |
|---|--------------------------------------|
| GL.LIGHT lx,ly,lz | Directional light (auto-normalised) |
| GL.AMBIENT a | Ambient level 0.0-1.0 (default 0.25) |
| GL.POINTLIGHT x,y,z | White point light |
| GL.POINTLIGHT x,y,z,r,g,b,intensity,dist | Full coloured point light |
| GL.LIGHTSOFF | Remove all point lights |

Material Properties

| Command / Syntax | Description |
|-----------------------|---------------------------------------|
| GL.COLOUR r,g,b | Base colour (0-255 each) |
| GL.SHINE n | Specular shininess 0-200 (default 30) |
| GL.ALPHA n | Opacity 0.0-1.0 (default 1.0) |
| GL.EMISSIVE r,g,b | Self-glow colour (0-255 each) |
| GL.WIRECOLOR r,g,b | Edge colour for SOLIDWIRE mode |
| GL.FOG r,g,b,near,far | Linear depth fog |
| GL.FOGOFF | Disable fog |

Building Meshes

| Command / Syntax | Description |
|-------------------|--|
| GL.BEGIN | Start mesh definition |
| GL.VERTEX x,y,z | Add vertex (1-indexed within mesh) |
| GL.FACE i,j,k[,l] | Add tri or quad face (1-based indices). Winding order does not matter. |
| GL.END | Finalise mesh – assigns ID |

| | |
|---------------------------|-------------------------------------|
| GL.MESHID | Numeric var: ID of last GL.END mesh |
| GL.SPHERE r[,wSegs,hSegs] | Create a sphere mesh directly |

Textures & Effects

| Command / Syntax | Description |
|-----------------------------------|--|
| GL.TEXTURE id,name\$ [,repeat] | Apply image as texture. repeat = tile count (default 4). |
| GL.CHROME id[,roughness] | Real-time reflective chrome. roughness 0.0=mirror, 0.05=polish |

Transforming Meshes

| Command / Syntax | Description |
|-------------------------|----------------------------------|
| GL.TRANSLATE id, x,y,z | Set world position |
| GL.ROTATE id, rx,ry,rz | Rotation in degrees (X,Y,Z axes) |
| GL.SCALE id, sx,sy,sz | Scale on each axis |
| GL.DRAW id / GL.DRAWALL | Render one mesh / all meshes |

■ *Transforms are set per frame — not cumulative. Call GL.ROTATE every frame to animate.*

Demos

| Command / Syntax | Description |
|------------------|--|
| RUN GLDEMO | Spinning wireframe cube |
| RUN GL3D | Solid cube + pyramid |
| RUN GLSHADE | Phong shading, 3 objects, directional light |
| RUN SGIDEMO | SGI Necker cube logo (W=wire, S=solid) |
| RUN GLDEMO3 | Advanced: fog, chrome sphere, point lights, shadows, mouse orbit, textured floor |

13. Audio — SOUND & WAVE ★ New in v0.5

OSAWARE BASIC supports Amiga-compatible audio synthesis via the Web Audio API. Four independent voice channels can play simultaneously with custom waveforms, panned left or right.

SOUND — Play a tone through the speaker

| | |
|----------------|--|
| Syntax | SOUND frequency, duration [, volume [, voice]] |
| Params | frequency — 20 to 15000 Hz duration — in jiffies (1/60 second). 60 = 1 second. volume — 0 (silent) to 255 (full). Default 127. voice — 0..3. Voices 0,3 = left channel. 1,2 = right. Default 0. |
| Example | SOUND 440, 60 ' A4 for one second SOUND 261, 30, 200, 0 ' middle C, half second, loud, left |
| Idiom | <i>Duration is in jiffies matching the original Amiga BASIC specification. 60 jiffies = 1 second. Voices 0 and 3 pan left; voices 1 and 2 pan right.</i> |

SOUND WAIT / SOUND RESUME — Queue and synchronise sounds across voices

| | |
|----------------|---|
| Syntax | SOUND WAIT SOUND RESUME |
| Example | SOUND WAIT SOUND 261, 60, 160, 0 ' C — left SOUND 329, 60, 140, 1 ' E — right SOUND 392, 60, 140, 2 ' G — right SOUND RESUME |
| Idiom | <i>SOUND WAIT queues all subsequent SOUND statements. SOUND RESUME releases the queue, playing all sounds simultaneously. Essential for chords and multi-voice music.</i> |

WAVE — Define a custom waveform for a voice channel

| | |
|----------------|---|
| Syntax | WAVE voice, SIN WAVE voice, arrayName |
| Example | WAVE 0, SIN ' reset to sine (default) ' Sawtooth wave: DIM W(255) FOR I=0 TO 255 : W(I) = I - 128 : NEXT I WAVE 0, W ' Organ (additive harmonics): DIM W(255) FOR I=0 TO 255 A = I * 6.2832 / 256 W(I) = INT(60*SIN(A) + 30*SIN(2*A) + 20*SIN(3*A)) NEXT I WAVE 0, W |
| Idiom | <i>Array must have at least 256 elements, each in range -128 to 127. WAVE voice, SIN resets to clean sine. Use ERASE after WAVE to free array memory.</i> |

| Command / Syntax | Description |
|------------------|------------------|
| BEEP | Short alert tone |

| | |
|---------------------------|---|
| SOUND f,d [,v [,ch]] | Tone at frequency f for d jiffies, volume v, voice ch |
| SOUND WAIT / SOUND RESUME | Queue / play all queued sounds simultaneously |
| WAVE ch, SIN | Reset voice ch to sine wave |
| WAVE ch, arrayName | Set custom 256-element waveform for voice ch |

■ *Demo: RUN AUDIOTEST — 8 demos covering scales, chords, arpeggios, and waveform comparison.*


14. Subprograms — SUB & FUNCTION ★ New in v0.5

Subprograms let you build reusable, self-contained routines with their own local variables. Variables inside a SUB are local by default and do not affect the calling code unless explicitly declared SHARED.

SUB / END SUB — Define a subprogram with local variable scope

| | |
|----------------|---|
| Syntax | <pre>SUB name[(param1, param2, ...)] STATIC ...body... END SUB</pre> |
| Example | <pre>10 A = 3 : B = 4 : C = 0 20 CALL AddNums(A, B, C) 30 PRINT "Sum = "; C ' prints 7 40 END 1000 SUB AddNums(X, Y, Z) STATIC 1010 Z = X + Y 1020 END SUB</pre> |
| Idiom | <i>The STATIC keyword is required on the declaration line. Place SUB definitions after END in the main program. Variables inside a SUB are local — they do not affect the main program.</i> |

CALL — Call a subprogram

| | |
|---------------|---|
| Syntax | <pre>CALL name(arg1, arg2) name arg1, arg2 ' implicit call, no CALL needed</pre> |
| Idiom |  Each CALL must be on its own line. CALL redirects execution flow and any colon-separated statements after it on the same line will be lost. |

Pass by Reference / Pass by Value — How parameters are transferred

| | |
|-----------------|--|
| By ref | <pre>CALL Sub(A, B) ' changes to A,B write back to caller</pre> |
| By value | <pre>CALL Sub((A), (B)) ' A,B protected, wrapped in extra parens</pre> |
| Idiom | <i>Parameters are passed by reference by default. Modifying a parameter inside the SUB modifies the caller's variable. Wrap in extra parentheses to pass by value.</i> |

SHARED — Share main-program variables inside a SUB

| | |
|----------------|---|
| Syntax | <pre>SHARED var1, var2 (inside SUB body)</pre> |
| Example | <pre>10 Score = 0 20 CALL AddPoint 30 PRINT Score ' prints 1 40 END 1000 SUB AddPoint STATIC 1010 SHARED Score 1020 Score = Score + 1 1030 END SUB</pre> |
| Idiom | <i>Any variable listed in SHARED is connected to the main program. Changes inside the SUB write back on exit. All other variables remain local.</i> |

FUNCTION / END FUNCTION — Define a subprogram that returns a value

| | |
|----------------|---|
| Syntax | <pre>FUNCTION name[(params)] STATIC name = returnValue END FUNCTION</pre> |
| Example | <pre>10 PRINT Square(5) ' prints 25 20 END 1000 FUNCTION Square(N) STATIC 1010 Square = N * N 1020 END FUNCTION</pre> |

| Command / Syntax | Description |
|---|---------------------------------------|
| SUB name(p) STATIC ... END SUB | Define subprogram with local scope |
| CALL name(args) | Call subprogram (one CALL per line) |
| name args | Implicit call (no CALL keyword) |
| SHARED var1, var2 | Share main-program vars inside SUB |
| STATIC var1 | Persist specific locals between calls |
| EXIT SUB / EXIT FUNCTION | Return early |
| FUNCTION name(p) STATIC | Define function that returns a value |
| ■ <i>Demo: RUN TESTS — K17 tests SUB, SHARED, and STATIC behaviour.</i> | |

15. Networking — WebSocket & SSH

| Command / Syntax | Description |
|------------------|---|
| WS.OPEN url\$ | Open connection (pauses until open/error) |
| WS.SEND msg\$ | Send text message |
| WS.RECV\$ | Pop next message (" " if empty) |
| WS.STATUS | 0=closed 1=connecting 2=open 3=error |
| WS.CLOSE | Close connection |
| WS.CLEAR | Flush message queue |
| WS.ONMSG lbl | GOSUB lbl on each incoming message |

```
10 WS.OPEN "ws://myserver:5000"  
20 IF WS.STATUS<>2 THEN PRINT "Failed":END  
30 WS.SEND "hello"  
40 SLEEP 500 : PRINT WS.RECV$  
50 WS.CLOSE
```

SSH Terminal — RUN SSH

Connects to SSH via a WebSocket bridge server. Prompts for bridge URL, SSH host, username and password.

```
npm install -g wssh  
wssh --port 5000 --host 0.0.0.0
```

16. AI Commands

| Command / Syntax | Description |
|--------------------|--|
| AIKEY "sk-ant-..." | Set API key (session only, never saved) |
| AIKEY | Prompt for API key with masked input |
| AI "prompt" | Send prompt, stream response to terminal |
| AI "prompt", R\$ | Send prompt, store in R\$ |
| AINUM "prompt", N | Send prompt, store numeric result in N |
| AICLEAR | Clear conversation history |

```
AI "Tell me a joke"
```

```
AI "Capital of France?", A$ : PRINT A$
```

```
AINUM "Rate this 1-10: flying cars", N : PRINT N
```

17. System Commands

| Command / Syntax | Description |
|---------------------|---|
| LOAD filename | Load from virtual filesystem |
| RUN [filename] | Run current or named program |
| SAVE filename | Save to browser storage |
| SAVE DOWNLOAD name | Download current program as name.bas file |
| NEW | Clear program from memory |
| LIST [n[-m]] | List program lines |
| EDIT n | Edit line n |
| DELETE n[-m] | Delete line(s) |
| MERGE filename | Merge into current program |
| TRON / TROFF | Enable/disable line trace |
| MEM / INFO | Program info: line count, highest line, variables |
| HISTORY | Show command history |
| BEEP | Short beep |
| RANDOMIZE [TIMER] | Seed RNG |
| FULLSCREEN [ON OFF] | Expand terminal to fill browser |
| OVERSCAN [ON OFF] | Full viewport, hide all chrome |
| LABELS | List all alphanumeric labels in current program |
| INKEY | KeyCode of last key pressed (0=none) |
| GETKEY() | Wait for keypress, return keyCode |
| LOCATE row,col | Position text cursor |
| CTRL+C / CTRL+Z | Break running program |

Built-in variables

| Command / Syntax | Description |
|------------------|--|
| WIDTH / HEIGHT | Canvas dimensions in pixels |
| COLS / ROWS | Terminal size in characters |
| DATE\$ / TIME\$ | Current date / time strings |
| UPTIME | Seconds since interpreter started |
| INKEY | Last key code |
| LINES / MAXLINE | Count / highest line number in program |
| ERL / ERR | Line number / code of last error |
| WS.STATUS | WebSocket status (0-3) |
| WS.RECV\$ | Next queued WebSocket message |

18. Colour Palette

Use COLOUR n for text, or pass directly to graphics commands as the colour argument.

| | | | |
|----------------------|-------------------|--------------------|---------------------|
| 0 White | 1 Lavender | 2 Red | 3 Green |
| 4 Yellow | 5 Magenta | 6 Cyan | 7 Lt Grey |
| 8 Dk Blue | 9 Dk Red | 10 Dk Green | 11 Lt Yellow |
| 12 Dk Magenta | 13 Teal | 14 Gold | 15 Grey |
| 16 Black | | | |

■ Colour 3 (green) is the classic terminal colour. Colour 16 (black) erases graphics.

19. Quick Reference Card

| Command / Syntax | Description |
|---|---|
| PRINT x;y | Print values; ; suppresses space/newline |
| INPUT "p";V | Prompt user, store in V |
| GOTO n / GOSUB n / RETURN | Jump / call subroutine / return |
| IF c THEN s [ELSE t] | Conditional |
| FOR I=a TO b STEP s / NEXT I | Count loop |
| WHILE c / WEND | Condition loop |
| END / STOP / CONT | Stop / pause / continue |
| CLS / COLOUR n | Clear screen / set colour (1-16) |
| PSET x,y,c / LINE x1,y1,x2,y2,c | Draw pixel / line |
| CIRCLE x,y,r,c / FILLRECT x1,y1,x2,y2,c | Circle / filled rect |
| PAINT x,y,c | Flood fill |
| DELAY ms / SLEEP ms | Engine delay / pause once |
| MOUSE ON/OFF ON MOUSE GOSUB lbl | Enable mouse / set handler |
| MOUSE(0)..MOUSE(6) | Button state and cursor position |
| OBJECT.SHAPE id,def\$ OBJECT.X/Y id,v | Sprite shape / move |
| OBJECT.ON/OFF [id] OBJECT.START/STOP | Show/hide / animate |
| GL.INIT GL.CLS GL.PERSPECTIVE fov | 3D init / clear / FOV |
| GL.CAMERA x,y,z GL.LOOKAT x,y,z | Camera position / target |
| GL.WIRE / GL.SOLID / GL.SOLIDWIRE | Render mode |
| GL.LIGHT lx,ly,lz GL.AMBIENT a | Directional / ambient light |
| GL.POINTLIGHT x,y,z[,r,g,b,i,d] | Point light |
| GL.COLOUR r,g,b GL.SHINE n GL.ALPHA n | Material colour/shine/opacity |
| GL.FOG r,g,b,near,far GL.FOGOFF | Fog on / off |
| GL.BEGIN..GL.VERTEX..GL.FACE..GL.END | Build mesh |
| GL.SPHERE r[,w,h] GL.CHROME id[,rough] | Sphere / chrome reflection |
| GL.TEXTURE id,name\$[,rep] | Apply texture |
| GL.TRANSLATE/ROTATE/SCALE id,... | Transform mesh |
| GL.DRAWALL | Render all meshes |
| LOADIMG "n","url" DISPLAY "n"[,x,y,w,h] | Store / draw image |
| ★ SOUND f,d[,v[,ch]] | Tone: freq, jiffies, vol 0-255, voice 0-3 |

| | |
|---------------------------------------|--|
| ★ SOUND WAIT / SOUND RESUME | Queue / play sounds simultaneously |
| ★ WAVE ch, SIN arr | Set waveform for voice (SIN or 256-elem array) |
| ★ SUB name(p) STATIC ... END SUB | Define subprogram with local scope |
| ★ CALL name(args) | Call subprogram (one CALL per line) |
| ★ SHARED var1,var2 | Share main-program vars inside SUB |
| WS.OPEN url\$ WS.SEND msg\$ WS.RECV\$ | WebSocket connect/send/receive |
| AI "prompt"[,R\$] AINUM "p",N | Ask Claude AI |
| LOAD/RUN/SAVE/LIST/MEM | Program management |
| ★ SAVE DOWNLOAD name | Download as name.bas file |
| ★ FULLSCREEN [ON OFF] | Full browser window |
| ★ LABELS | List all defined labels |
| LINES / MAXLINE DATE\$ / TIME\$ | System variables |

★ = New in v0.5 Demos: [RUN AUDIOTEST](#) [RUN TESTS](#) [RUN MENU](#)