

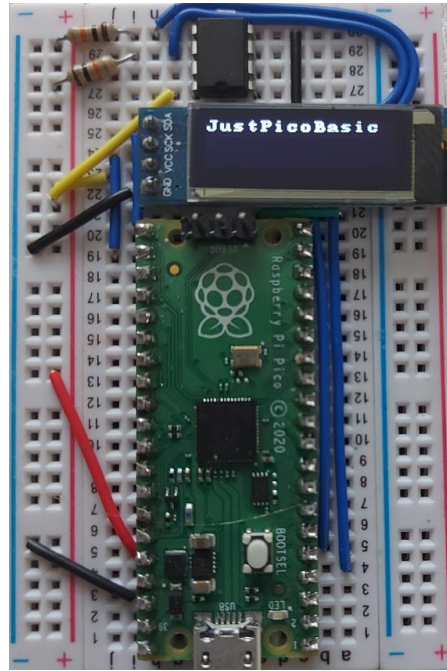
#JustPicoBasic manual

The best way to see how it works is to run a few examples available at:

<https://github.com/bgolab/JustBasic/tree/main/examples>

Wiring

Hardware components: **RPI PICO**, **OLED 0.91" I2C**, **EEPROM 24c64 I2C**, **2x resistors 10kohm**



Getting started

Power the PICO through the USB. Start a terminal emulator e.g. **Putty**, **TeraTerm**. Use the following settings: **9600,8,N,1**. Copy & Paste is supported. Copy the current UF2 file (e.g. **JustBasic-1.0b36.uf2**) to the PICO disk. The PICO will boot. Enter '?' and press ENTER. You will see available SYSTEM & BASIC commands.

```
COM10 - Tera Term VT
File Edit Setup Control Window Help
?
JustPicoBasic v1.0B40
(C) 2021 bg
HW: RP2040

MAX: SRC/LINE:8000/160, HEAP:1000, NAMES/LENGTH:100/8, IF/LOOP:50/50

SYSTEM: ?[info], l[load], s[save], c[code], r[run], n[new], b[bye], t[tech], @n[], t0-t2

BASIC: print, input, inkey, cls, data, read, restore, if, then, else, endif, for, to, step, break, next, while, endwh
ile, goto, gosub, return, rem, peek, poke, pause, rnd, abs, sin, cos, exp, log, sqr, sgn, hex$, str$, chr$, asc, left
$, mid$, right$, len, val, int, fix, gettick, dim, pmode, dwrite, awrite, dread, aread, end, ,, ;, +, -, and, or, not
, *, /, %, (, ), <, <=, >, >=, =, <>, ==, !=, lplot, ldrawing, lcircle, lprint, at, lref, lcis, integer, float, string,
ts, sm,

CONST: INPUT, OUTPUT, PULLUP, PULLDOWN, ADC, PWM, TSENSOR, HIGH, LOW, ENABLE, DISABLE,

#
```

Enter an example BASIC program:

```
pmode 25,1
for k=1 to 5
    dwrite 25,1 pause 500
```

dwrite 25,0 pause 500

next k
end

Use 'c' to see the code, 'r' to run the code. The built-in LED will blink. The 'n' clears the program memory.

You can use @N (e.g. @0) to delete Nth line or @N <code> (@0 print 1) to insert the line of code.

You can freely format the code – can put many commands in single line, etc. Capital and small letters accepted for keyword (i.e. 'CLS' and 'cls' are the same). Names recognizes small and capital letters (i.e. 'as' is different than 'AS')

Program structure

Program Structure	Description/Comments	Example
<code>code gosub subr1 end subr1: [subr1 code] return</code>	Command 'end' has to follow the last line of the main code. Subroutines have to follow the 'end'.	<code>print "Hi!" gosub callme end callme: print "Hi!" return</code>

System commands

Command	Description/Comments	Example
ESC key	Break the program while Running or prevent from Loading.	
?	Shows info about the VM (ver, available commands)	
c, cc	c-show the code w/ line numbers, cc-show the code w/o line numbers	
r	Run – run code from SRAM memory	
r <code>	Run - run single line of code NOT stored in memory (ad-hoc)	<code>r for i=1 to 5 print i, ", ", i*i next i end</code>
n	New – clear VM memory and code	
b	Bye: PICO – reboot VM in disk mode, Windows - exist	
l	l - Load program from EEPROM (auto.bas)	
s	s - Save program to EEPROM (auto.bas)	
ee	ee - EEPROM erase – now command disabled	
ed	ed - EEPROM dump – show EEPROM content	
is	I2C scan – show I2C devices on both I2C buses	
t0 t1 t2	Program Flow Tracing: t0 – OFF; t1 - Step Mode; t2 – Cont Mode	<code>T1 T2 - enable tracing mode</code>
@N	Pico Editor: @N - delete N-th line	<code>@3 – removes 3rd line of code</code>
@N <code>	Pico Editor: @N <code> - insert <code> before N-th line	<code>@4 CLS – inserts 'CLS' before line 4</code>

The BASIC language

Command	Description/Comments	Example
<code>SM <entity> enable/disable</code>	System Mode Configuration. Entity: ESC (default=enabled) - ESC key check (disable to boost performance) OLED (default=enabled) (NOTE: OLED auto-detection is supported now)	<code>SM ESC disable SM OLED 1</code>
REM	Comment	<code>REM MyFunc</code>
CLS	Clear Screen	<code>CLS</code>

VARIABLES & EXPRESSIONS

Suffix-based (suffix #, \$, OR no suffix to declare variable type) varname and array syntax

-variable name: up to 8chars letter&digits starting w/ a letter (digits, '#', '\$', '_', ':' accepted)

-expr(arithmetic expression): combination of INT/FLOAT and ops/brackets (+, -, *, /, %, (,)) and INT/FLOAT vars;

-sexpr(string expression): combination of string, string functions (with suffix \$) and string vars (with suffix \$) and '+' op

-variable type differentiation through the suffix (no suffix – integer, '#' suffix – float, '\$' suffix – string

-array index counts from 0 (for: DIM a(3) available array elements are referred by a(0), a(1), a(2)

-string arrays must have 2-dimensions e.g. DIM a\$(2,5) – 2 strings, maximum length of a string is 5 characters; string arrays are always referred through single index a\$(0), a\$(1) for DIM a\$(2,5) array

Command	Description/Comments	Example
<code>var=expr</code>	INT var, name=expr, 1 st -reference creates var(value=0);	<code>sy=2*abs(-15) + a*20</code>
<code>var#=expr</code>	FLOAT var (# suffix), name#=expr, 1 st -reference creates var w/ value=0	<code>w#=2*a#+abs(-1.0)</code>
<code>var\$=sexpr</code>	STR var (\$ suffix), name\$=sexpr	<code>v\$=a\$+left\$(str\$(13),1)</code>
<code>DIM var(s1[,s2]), var#(s1[,s2]), var\$(s1,s2)</code>	INT/FLOAT/STRING array, 1/2-dimensions; multi-array declaration (array names separated by comma)	<code>DIM a(3), b#(4,4), c\$(4,5)</code>
<code>var(expr[,expr])=expr</code>	INT: name(item)=expr	<code>a(0)=3</code>
<code>var#(expr[,exp])=expr</code>	FLOAT: name#(item)=expr	<code>b#(0)=2.5</code>
<code>var\$(expr)=sexpr</code>	STRING: name\$(item)=sexpr	<code>c\$(2)="abc"</code>

CONSTANTS

Command	Description/Comments	Example
<code>ENABLE (1), DISABLE (0),</code>	Generic constants	<code>sm esc disable</code>
<code>HIGH (1), LOW (0)</code>	Generic constants	<code>dwrite 12,low</code>
<code>INPUT, OUTPUT, PULLUP, PULLDOWN, ADC, PWM, TSENSOR</code>	GPIO constants usefull for hw GPIO modes	<code>pmode 25,output dwrite 25,low</code>

Suffixless varname and array syntax

- variable needs to be declared (if not declared integer type is assumed), multi-declaration in single command supported
- initialization during the declaration phase is not supported yet
- array index counts from 0 (for: DIM a(3) available array elements are referred by a(0), a(1), a(2))
- string arrays must have 2-dimensions e.g. string a(2,5) – 2 strings, maximum length of a string is 5 characters; string arrays are always referred through single index a(0), a(1) for string a(2,5) array

Command	Description/Comments	Example
<code>integer vname, vname2(s1[,s2]),...</code>	Declare integer var or 1/2-dimensional array; value=0 set	<code>integer a, b(8,2) b(0,0)=1</code>
<code>float vname, vname2(s1[,s2]),...</code>	Declare float var or 1/2-dimensional array; value=0 set	<code>float c, b(0,0)=1.0</code>
<code>string vname, vname2(s1,s2),...</code>	Declare string var or 1/2-dimensional array; value=null set	<code>string b(2,4) a(1)="no"</code>

PROGRAM FLOW CONTROL

- cond: logical expression e.g. a>5 and b<10
- lexpr(logical expression): combination of conditional(<>,<,>=,<=,<=,<=,<=,<=,<=,<=,<=) and logical operators(AND,OR,NOT)
- NOTE: '<>' is equivalent to '!=', '<=' is equivalent to '<='

Command	Description/Comments	Example
<code>FOR v=expr TO expr [STEP expr] [code] NEXT var</code>	if STEP[default=1] is negative var decreases; INT/FLOAT supported; nesting supported	<code>FOR i = 5 TO 1 STEP -1 NEXT i END</code>
<code>WHILE lexpr [code] ENDWHILE</code>	INT/FLOAT supported; nesting supported; cond: AND/OR/NOT supported;	<code>a=0 while a<5 print a a=a+1 endwhile end</code>
<code>BREAK</code>	FOR and WHILE loops supported	<code>while 1>0 k=inkey() if(k!=0) then print k endif if(k==113) then break endif endwhile</code>
<code>IF lexpr THEN [code] [ELSE] [code] ENDIF</code>	INT/FLOAT supported; nesting supported AND/OR/NOT supported;	<code>if a>1 and b#>3.4 then print "ok" else print "bad" endif</code>
<code>label: GOTO label</code>	Label name starts with a letter, terminated by colon; up to 8 letter & digits(plus '_');	<code>k=1 again: print k k=k+1 if k<5 then goto again: endif</code>
<code>GOSUB label</code>	Label must be located after END	<code>gosub task0 end</code>
<code>label: RETURN</code>	label: [code] RETURN	<code>task0: print "done" return</code>
<code>END</code>	Last instruction. GOSUB labels follows END	

INPUT, OUTPUT, DATA

Command	Description/Comments	Example
---------	----------------------	---------

<code>PRINT expr[, sexpr], [;]</code>	Prints <code>expr</code> , <code>sexpr</code> separated by <code>‘</code> ;’ <code>‘</code> ;’ to skip NEW LINE	<code>PRINT "6/3=", 6/3 (with NEW LINE)</code> <code>PRINT 1; (w/o NEW LINE because of ‘;’)</code>
<code>INPUT var,...</code>	Assign int/float/str values to (array) <code>var</code>	<code>INPUT a(2), d#, n\$ print a(2), d#, n\$</code>
<code>DATA expr, sexpr;</code>	INT/FLOAT/STR supported	<code>DATA 1.5, 2*a</code>
<code>READ a, b#, d\$</code>	Assign DATA specified input to vars	<code>READ v, v#, v(), v#();</code>
<code>RESTORE</code>	Reset data pointer	

BUILT-IN FUNCTIONS

Command	Description/Comments	Example
<code>LEFT\$(sexpr, expr)</code>	Left part of the string	<code>k\$=LEFT\$("abc", 2) + "123" i=12</code>
<code>RIGHT\$(sexpr, expr)</code>	Right part of the string	
<code>MID\$(sexpr,expr,expr)</code>	Middle of the string	<code>i\$=MID\$(STR\$(i),2,3) PRINT i\$</code>
<code>HEX\$(expr)</code>	Hex\$(expr to hex string)	<code>PRINT HEX\$(NOT(0x0F))</code>
<code>STR\$(expr)</code>	Str\$(expr to string)	
<code>CHR\$(expr)</code>	Chr\$(expr%256 to ascii e.g. 65 to ‘A’)	<code>a=65 d\$=chr\$(a)</code>
<code>LEN(sexpr)</code>	LEN(string length)	<code>PRINT LEN("1234")->4</code>
<code>VAL(sexpr)</code>	Val(string to value)	<code>PRINT VAL("-1234")+1->-1233</code>
<code>ASC(sexpr)</code>	ASC(ascii code of the 1 st char of the string)	<code>PRINT ASC("AB")->65</code>
<code>SIN(expr)</code>	Sine	<code>PRINT "SIN:", SIN(3.14/6)</code>
<code>COS(expr)</code>	Cosine	<code>PRINT "COS:", COS(3.14/6)</code>
<code>SQR(expr)</code>	Square root	<code>PRINT "SQRT:", SQR(5)</code>
<code>EXP(expr)</code>	Exponential	<code>PRINT "EXP:", EXP(1)</code>
<code>LOG(expr)</code>	Logarithm	<code>PRINT "LOG:", LOG(2.71)</code>
<code>SGN(expr)</code>	Sign	<code>PRINT "SGN:", SGN(-5)</code>
<code>ABS(expr)</code>	Absolute	<code>PRINT "ABS:", ABS(-5)</code>
<code>RND(max)</code>	Hw-based rnd generator + von Neuman whitenizer	<code>PRINT "RND: ", RND(1000)</code>
<code>GETTICK()</code>	Tick number	<code>a=gettick()</code>
<code>PAUSE msec</code>	Delay (blocking) in msec	<code>PAUSE 2*500</code>
<code>INKEY()</code>	Pressed key, OR 0; non-blocking (no-waiting)	
<code>INT(expr)</code>	QBASIC like	<code>a=INT(1.1) b=INT(-1.1) PRINT a, ", ", b (1,-2)</code>
<code>FIX(expr)</code>		<code>c=FIX(1.9) d=FIX(-1.9) PRINT c, ", ", d (1,-1)</code>
<code>AND(expr,expr)</code>		<code>PRINT AND(0x3,0xF)</code>
<code>OR(expr,expr)</code>		<code>PRINT OR(0x3,0xF)</code>
<code>NOT(expr)</code>		<code>PRINT HEX\$(NOT(0x0F))</code>

PICO HARDWARE SUPPORT

Command	Description/Comments	Example
<code>POKE addr, value</code>	Memory write; hex supported	<code>REM SYSTICK (ST)</code> <code>STCSR=0xe000e010</code> <code>STRVR=0xe000e014</code> <code>STCVR=0xe000e018</code> <code>poke STCSR,0 poke STRVR,0x1e847 poke STCSR,5</code> <code>for k=1 to 50</code> <code>print and(peek(STCVR), 0x0FFFFFFF) pause 1000</code> <code>next k</code>
<code>PEEK(addr)</code>	Memory read; hex supported	
<code>PMODE pin, mode</code>	m:0-IN,1-OUT,2-PULLUP,3-PULLDOWN, 10-ADC, 15-PWM, 20-TSENSOR	<i>NOTE: As of 1.0b39 you can use HW CONSTANTS</i> <code>pmode 100, tsensor temp= aread(100)</code>
<code>AREAD(pin)</code>	Read analog pin; pins=26-29 – analog; pin=100 – temperature virtual pin	<code>pmode 26, adc voltage=aread(26)</code> <code>pmode 100,tsensor temp= aread(100)</code>
<code>AWRITE pin, cycles</code>	PWM duty=cycles/65535 cycle: 0-65535)	<code>pmode 22, pwm awrite 22, 16000</code>
<code>DREAD(pin)</code>	Read digital pin	<code>y=15 pmode y, input pmode y, pullup</code>

		<i>for k=1 to 2 step 0 pause 50 print dread(y) next k</i>
<i>DWRITE pin, value</i>	<i>Write digital pin</i>	<i>pmode 25,output dwrite 25,high pause 3000 dwrite 25,low</i>

GRAPHIC LCD/OLED SUPPORT (currently: OLED0.91 support)

Command	Description/Comments	Example
<i>SM OLED 1</i>	<i>Enable OLED support</i>	<i>NOTE: As of 1.0b31 OLED auto-detection is supported</i>
<i>LPLOT X, Y</i>	<i>Draw point at X, Y</i>	<i>for x=0 to 127 lplot x,fix(15+15*sin(6.28*x/128)) next x lref</i>
<i>LDRAW X0, Y0, X1, Y1</i>	<i>Draw line from X0, Y0 to X1, Y1</i>	<i>ldraw 10, 10, 20,20 lref</i>
<i>LCIRCLE x, y, r</i>	<i>Draw circle at x, y, r</i>	<i>lcircle 15, 15, 10</i>
<i>LPRINT expr, sexpr [AT x,y]</i>	<i>Print expr, sexpr separated by ',' [AT x, y] (default 0,0)</i>	<i>lprint "2+2=", 2+2 AT 10,10 lref</i>
<i>LCLS</i>	<i>Clear Screen (actually the buffer)</i>	<i>lcls lref</i>
<i>LREF</i>	<i>Refresh LCD (copy mem to LCD)</i>	<i>lref</i>