
```
================================================================================================

 PICO-8 v0.1.11c
 http://www.pico-8.com
 (c) Copyright 2014-2017 Lexaloffle Games LLP
 Author: Joseph White // hey@lexaloffle.com

 PICO-8 is built with:
        SDL2 http://www.libsdl.org
        Lua 5.2 http://www.lua.org  // see license.txt
        GIFLIB http://giflib.sourceforge.net/
        WiringPi http://wiringpi.com/

================================================================================================

Welcome to PICO-8!

        PICO-8 is a fantasy console for making, sharing and playing tiny games and other computer
        programs. When you turn it on, the machine greets you with a shell for typing in Lua programs
        and provides simple built-in tools for creating sprites, maps and sound.

        The harsh limitations of PICO-8 are carefully chosen to be fun to work with, encourage small
        but expressive designs and hopefully to give PICO-8 cartridges their own particular look and
        feel.


:: Keys

        Toggle Fullscreen: Alt+Enter
        Quit: Alt+F4 or command-Q
        Reload/Run/Restart cart: Ctrl+R
        Quick-Save: Ctrl+S
        Mute/Unmute: Ctrl+M
        Player 1 defaults: Cursors + ZX / NM / CV
        Player 2 defaults: SDFE + tab,Q / shift A
        Enter or P for pause menu (while running)
        // use KEYCONFIG to change the defaults.


:: Specs

        Display: 128x128, fixed 16 colour palette
        Input: 6 button controllers
        Cartridge size: 32k
        Sound: 4 channel, 64 definable chip blerps
        Code: Lua (max 8192 tokens of code)
        Sprites: Single bank of 128 8x8 sprites (+128 shared)
        Map: 128x32 8-bit cels (+128x32 shared)


:: Hello World

        After PICO-8 boots, try typing some of these commands followed by enter:
```

```
        PRINT("HELLO WORLD")
        RECTFILL(80,80,120,100,12)
        CIRCFILL(70,90,20,14)
        FOR I=1,4 DO PRINT(I) END
```

(Note: PICO-8 only displays upper-case characters -- just type normally without capslock!)

You can build up an interactive program by using commands like this in the code editing mode along with two special callback functions _UPDATE and _DRAW. For example, the following program allows you to move a circle around with the cursor keys. Press escape to switch to the code editor and type or copy & paste the following code:

```
X = 64  Y = 64
FUNCTION _UPDATE()
 IF (BTN(0)) THEN X=X-1 END
 IF (BTN(1)) THEN X=X+1 END
 IF (BTN(2)) THEN Y=Y-1 END
 IF (BTN(3)) THEN Y=Y+1 END
END

FUNCTION _DRAW()
 RECTFILL(0,0,127,127,5)
 CIRCFILL(X,Y,7,8)
END
```

Now press escape to return to the console and type RUN (or press CTRL-R) to see it in action. See the example cartridges for more complex programs.

If you want to store your program for later, use the SAVE command:

```
> SAVE REDCIRC
```

:: Example Cartridges

These cartridges are included with PICO-8 and can be installed by typing:

```
INSTALL_DEMOS
CD DEMOS
```

```
HELLO      Greetings from PICO-8
API        Demonstrates most PICO-8 functions
JELPI      Platform game demo w/ 2p support
CAST       2.5D Raycaster demo
MANDEL     Mandelbrot explorer
COLLIDE    Example wall and actor collisions
BUTTERFLY  Serpinsky triangles thing
DRIPPY     Draw a drippy squiggle
STOMPY     Music cart
```

To run a cartridge, open PICO-8 and type:

```
LOAD JELPI
RUN
```

Press escape to stop the program, and once more to enter editing mode.

:: File System

These commands can be used to manage files and directories (folders):

```
DIR         list the current directory
CD BLAH     change directory
CD ..       go up a directory
CD /        change back to top directory (on pico-8's virtual drive)
MKDIR BLAH  make a directory
FOLDER      open the current directory in the host operating system's file browser

LOAD BLAH  load a cart from the current directory
SAVE BLAH  save a cart to the current directory
```

If you want to move files around, duplicate them or delete them, use the FOLDER command and do it in the host operating system.

```
        The default location for PICO-8's drive is:

        Windows: C:/Users/Yourname/AppData/Roaming/pico-8/carts
        OSX: /Users/Yourname/Library/Application Support/pico-8/carts
        Linux: ~/.lexaloffle/pico-8/carts

        You can change this and other settings in pico-8/config.txt

        Tip: The drive directory can be mapped to a cloud drive (provided by Dropbox, Google
        Drive or similar) in order to provide a single disk shared between PICO-8 machines
        spread across different host machines.


:: Loading and Saving

        When using LOAD and SAVE, the .P8 extention can be omitted and is added automatically.

        Saving to a .p8.png extention will save the cartridge in a special image format that
        looks like a cartridge.

        Use a filename of "@CLIP" to load or save to the clipboard.

        Once a cartridge has been loaded or saved, it can also be quick-saved with CTRL-S

        :: Saving .p8.png carts with a text label and preview image

                To generate a label image saved with the cart, run the program first and press F7 to grab
                whatever is on the screen. The first two lines of the program starting with '--' are also drawn
                to the cart's label.

                e.g.
                -- OCEAN DIVER LEGENDS
                -- BY LOOPY


        :: Code size restrictions for .png format

                When saving in .png format, the compressed size of the code must be less than 15360 bytes. To find
                out the current size of your code, use the INFO command. The compressed size limit is not enforced
                for saving in .p8 format.



:: Backups

        If you quit without saving changes, or overwrite an existing file, a backup of the
        cartridge is saved to pico-8/backup.


:: Configuration

        :: config.txt

                You can find some settings in config.txt. Edit the file when PICO-8 is not running.

                Windows: C:/Users/Yourname/AppData/Roaming/pico-8/config.txt
                OSX: /Users/Yourname/Library/Application Support/pico-8/config.txt
                Linux: ~/.lexaloffle/pico-8/config.txt

                Use the -home switch (below) to use a different path to store config.txt and other data.


        :: Commandline parameters

                // note: these override settings found in config.txt

                pico-8 [switches] [filename.p8]

                -width n                set the window width
                -height n               set the window height
                -windowed n             set windowed mode off (0) or on (1)
                -sound n                sound volume 0..256
                -music n                sound volume 0..256
                -joystick n             joystick controls starts at player n (0..7)
                -pixel_perfect n        1 for unfiltered screen stretching at integer scales (on by default)
```

```
-draw_rect x,y,w,h       absolute window coordinates and size to draw pico-8's screen
-run filename            load and run a cartridge
-x filename                  execute a PICO-8 cart headless and then quit
-p param_str             pass a parameter string to the specified cartridge
-splore                  boot in splore mode
-home path               set the path to store config.txt and other user data files
-desktop path            set a location for screenshots and gifs to be saved
-screenshot_scale n      scale of screenshots.  default: 3 (368x368 pixels)
-gif_scale n             scale of gif captures. default: 2 (256x256 pixels)
-gif_len n               set the maximum gif length in seconds (1..120)
-gui_theme n             use 1 for a higher contrast editor colour scheme
-timeout n               how many seconds to wait before downloads timeout (default: 30)
-software_blit n         use software blitting mode off (0) or on (1)
-foreground_sleep_ms n   how many milliseconds to sleep between frames.
-background_sleep_ms n   how many milliseconds to sleep between frames when running in background
```

:: Controller Setup

PICO-8 uses the SDL2 controller configuration scheme. It will detect common controllers on startup and also looks for custom mappings in sdl_controllers.txt in the same directory as config.txt. sdl_controllers.txt has one mapping per line.

To generate a custom mapping string for your controller, use either the controllermap program that comes with SDL2, or try http://www.generalarcade.com/gamepadtool/

To set up which keyboard keys trigger joystick buttons presses, use KEYCONFIG.

:: Screenshots, Videos and Cartridge Labels

While a cartridge is running use:

```
F6 Save a screenshot to desktop
F7 Capture cartridge label image
F8 Start recording a video
F9 Save GIF video to desktop (max: 8 seconds by default)

// if F6..F9 are not available on your system, use F1..F4 or CTRL-6..9
```

You can save a video at any time (it is always recording) -- use F8 just to reset the video starting point if you want something less than 8 seconds long.

To change the maximum gif length, edit gif_len in config.txt to specify the number of seconds to record for. The gif format can not match 30fps exactly, so PICO-8 instead uses the closest match: 33.3fps.

:: Sharing Cartridges

There are three ways to share carts made in PICO-8:

1. Share the .p8 or .p8.png file directly with other PICO-8 users

   Type FOLDER to open the current folder in your host operating system.

2. Post the cart on the Lexaloffe BBS to get a web-playable version

   http://www.lexaloffle.com/pico-8.php?page=submit

3. Export the cartridge to a stand-alone html5 or native binary player
   (see the exporters section for details)

:: Exporters / Importers

The EXPORT command can be used to generate png, wav files and stand-alone html5 and native binary cartridge players. The output format is inferred from the filename extention (e.g. .png).

You are free to distribute and use exported cartridges and data as you please, provided that you have permission from the author and contributors.

```
:: Sprite Sheet (.png)

        IMPORT BLAH.PNG     --  expects 128x128 png and colour-fits to the pico-8 palette
        EXPORT BLAH.PNG     --  use folder() to locate the exported png

:: SFX and Music (.wav)

        EXPORT BLAH.WAV     --  export music from the current pattern (when editor mode is MUSIC)
        EXPORT BLAH.WAV     --  export the current SFX (when editor mode is SFX)
        EXPORT BLAH%D.WAV   --  exports all of the SFXs as blah0.wav, blah1.wav .. blah63.wav

:: HTML Player (.html)

        To generate a stand-alone html player (foo.html and foo.js):
        > EXPORT FOO.HTML

        Or just the .js file:
        > EXPORT FOO.JS

        Optionally provide a custom html template with the -p switch:
        > EXPORT FOO.HTML -P ONE_BUTTON

        This will use the file {application data}/pico-8/plates/one_button.html as the
        html shell, replacing a special string, ##js_file##, with the .js filename.


:: Binary Player (.bin)

        To generate stand-alone executables for Windows, Linux (64-bit) and Mac OSX:

        > EXPORT FOO.BIN

        By default, the cartridge label is used as an icon with no transparency. To
        specificy an icon from the sprite sheet, use -i and optionally -s and/or -c
        to control the size and transparency.

                -I N  Icon index N with a default transparent colour of 0 (black).
                -S N  Size NxN sprites. Size 3 would be produce a 24x24 icon.
                -C N  Treat colour N as transparent. Use 16 for no transparency.

        For example, to use a 2x2 sprite starting at index 32 in the spritesheet,
        using colour 12 as transparent, and bundling an extra cartrige C0.P8:

                > EXPORT FOO.BIN -I 32 -S 2 -C 12


:: Exporting Multiple Cartridges

        Up to 16 cartridges can be bundled together by passing them to EXPORT, when generating
        stand-alone html5 or native binary players

        EXPORT FOO.HTML DAT1.P8 DAT2.P8 GAME2.P8

        During runtime, the extra carts can be accessed as if they were local files:

        RELOAD(0,0,0x2000, "DAT1.P8") -- load spritesheet from DAT1.P8
        LOAD("GAME2.P8") -- load and run another cart

        Currently only .p8 files are supported, and the filenames must be specified with
        the .p8 extension included.


:: Splore

        SPLORE is a built-in utility for browsing and organising both local and bbs (online)
        cartridges. Type SPLORE [enter] to launch it, or launch PICO-8 with -splore.

        It is possible to control SPLORE entirely with a joystick:
        LEFT and RIGHT to navigate lists of cartridges
        UP AND DOWN to select items in each list
        X,O or MENU to launch the cartridge

        While inside a cart, press MENU to favourite a cartridge or exit to splore.
        If you're using a keyboard, it's also possible to press F to favourite an item
        while it is selected in the cartridge list view.
```

When viewing a list of BBS carts, use the top list item to re-download a list of cartridges. If you are offline, the last downloaded list is displayed, and it is still possible to play any cartridges you have downloaded.

If you have installed PICO-8 on a machine with no internet access, you can also use INSTALL_GAMES to add a small selection of pre-installed BBS carts to your favourites list.

:: Quirks of PICO-8

Common gotchas to watch out for:

- The bottom half of the spritesheet and bottom half of the map occupy the same memory.
  // Best use only one or the other if you're unsure how this works.
- PICO-8 numbers only go up to 32767.99.
  // If you add 1 to a counter each frame, it will overflow after around 18 minutes!
- Lua arrays are 1-based by default, not 0-based. FOREACH starts at T[1], not T[0].
- cos() and sin() take 0..1 instead of 0..PI*2, and sin() is inverted.
- sgn(0) returns 1.
- Toggle fullscreen: use alt-enter on OSX (command-F is used for searching text).
- When you want to export a .png cartridge, use SAVE, not EXPORT. EXPORT will save only the spritesheet!

==========
Editor Modes
==========

Press escape to toggle between console and editor
Click editing mode tabs at top right to switch or press ALT+LEFT/RIGHT

** WARNING: The second half of the sprite sheet (banks 2 and 3), and the bottom half of the map share the same cartridge space. It's up to you how you use the data, but be aware that drawing on the second half of the sprite sheet could clobber data on the map and vice versa.

:: Code Editor

Hold shift to select (or click and drag with mouse)
CTRL-X, C, V to cut copy or paste selected
CTRL-Z, Y to undo, redo
CTRL-F to search for text in the current tab
CTRL-G to repeat the last search again
CTRL-L to jump to a line number
CTRL-UP, DOWN to jump to start or end
ALT-UP, DOWN to navigate to the previous, next function
CTRL-LEFT, RIGHT to jump by word
CTRL-D to duplicate current line
TAB to indent a selection (shift to un-indent)

To enter special characters that represent buttons, use SHIFT-L,R,U,D,O,X
Or press CTRL-K to toggle glyph mode

:: Tabs

Click the [+] button at the top to add a new tab.
Navigate tabs by left-clicking, or with ctrl-tab, shift-ctrl-tab.
To remove the last tab, delete any contents and then moving off it (CTRL-A, del, ctrl-tab)
When running a cart, a single program is generated by concatenating all tabs in order.

:: Code limits

The current number of code tokens is shown at the bottom right. One program can have a maximum of 8192 tokens. Each token is a word (e.g. variable name) or operator. Pairs of brackets, and strings each count as 1 token. commas, periods, LOCALs, semi-colons, ENDs, and comments are not counted.

Right click to toggle through other stats (character count, compressed size).
If a limit is reached, a warning light will flash. This can be disabled by right-clicking.

:: Sprite Editor

The sprite editor is designed to be used both for sprite-wise editing and for freeform pixel-level editing. The sprite navigator at the bottom of the screen provides an 8x8-wise view into the sprite-sheet, but it is possible to use freeform tools (pan, select) when dealing with larger or oddly sized areas.

Draw Tool
- Click and drag on the sprite to plot pixels
- Applies to visible area
- Hold CTRL to search and replace a colour
- Use right mouse button to select colour

Stamp Tool
- Click to stamp whatever is in the copy buffer
- Hold LCONTROL to treat colour 0 (black) as transparent

Select Tool // shortcut: LSHIFT or S
- Create a selection
- Enter or click to select none.

  If a pixel-wise selection is not present, many operations are instead applied to a sprite-wise selection. To select sprites, shift-drag in the sprite navigator.

Pan Tool // shortcut: space
- View the spritesheet.

Fill Tool
- Fill with the current colour
- Applies to the current selection
- If no selection, applies to visible area

Extra keys
- CTRL-Z to undo // single step only in 0.2.0
- CTRL-C to copy selected area or selected sprites
- CTRL-V to paste to current sprite location
- Q,W to switch to previous/next sprite
- 1,2 to switch to previous/next colour
- Tab to toggle fullscreen view
- Mousewheel or < and > to zoom (centered in fullscreen)

Operations on selected area or selected sprites:
- f to flip
- v to flip vertically
- r to rotate (must be square selection)
- Cursor keys to move (loops if sprite selection)

Sprite flags
- The 8 coloured circles are sprite flags for the current sprite.
- Each one can be true (on) or false (off), and are accessed by using the FSET and FGET functions. They are indexed from 0, from the left (0,1,2..7). See fset() for more information.

:: Map Editor

The pico-8 map is a 128x32 (or 128x64 using shared space) block of 8-bit values. Each value is shown in the editor as a reference to a sprite (0..255), but you can of course use the data to represent whatever you like.

The tools are similar to the ones used in sprite editing mode. Select a sprite and click and drag to paint values into the map.

To draw multiple sprites, select from sprite navigator with shift+drag
To copy a block of values, use the selection tool and then stamp tool to paste
To pan around the map, use the pan tool or hold space
Q,W to switch to previous/next sprite
Mousewheel or < and > to zoom (centered in fullscreen)

:: SFX Editor

There are 64 SFX ("sound effects") in a cartridge, used for both sound and music.

```
Each SFX has 32 notes, and each note has:
        A frequency   (C0..C5)
        An instrument (0..7)
        A volume      (0..7)
        An effect     (0..7)

Each SFX also has these properties:

        A play speed (SPD) : the number of 'ticks' to play each note for.
                // This means that 1 is fastest, 3 is 3x as slow, etc.

        Loop start and end : this is the note index to loop back and to
                // Looping is turned off when the start index >= end index

There are 2 modes for editing/viewing a SFX: Pitch mode (more suitable
for sound effects) and tracker mode (more suitable for music). The mode
can be changed using the top-left buttons, or toggled with TAB.


1. Pitch Mode

        Click and drag on the pitch area to set the frequency for each note,
        using the currently selected instrument (indicated by colour).

        Hold shift to apply only the selected instrument
        Hold CTRL to snap entered notes to the C minor pentatonic scale


2. Tracker Mode

        Each note shows: frequency octave instrument volume effect
        To enter a note, use q2w3er5t6y7ui zsxdcvgbhnjm (piano-like layout)
        Hold shift when entering a note to transpose -1 octave .. +1 octave
        New notes are given the selected instrument/effect values
        To delete a note, use backspace or set the volume to 0

        Click and then shift-click to select a range that can be copied
        (CTRL-C) and pasted (CTRL-V)

        Navigation:
                PAGEUP/DOWN or CTRL-UP/DOWN to skip up or down 4 notes
                HOME/END to jump to the first or last note
                CTRL-LEFT/RIGHT to jump across columns


3. Controls for both modes

        - + to navigate the current SFX
    < > to change the speed.
        SPACE to play/stop
        SHIFT-SPACE to play from the current group of 8
        A to release a looping sample
        Left click or right click to increase / decrease the SPD or LOOP values
         // Hold shift when clicking to increase / decrease by 4
         // Alternatively, click and drag left/right or up/down
        Shift-click an instrument, effect, or volume to apply to all notes.


:: Effects

        0 none
        1 slide          //  Slide to the next note and volume
        2 vibrato        //  Rapidly vary the pitch within one quarter-tone
        3 drop           //  Rapidly drop the frequency to very low values
        4 fade in        //  Ramp the volume up from 0
        5 fade out       //  Ramp the volume down to 0
        6 arpeggio fast  //  Iterate over groups of 4 notes at speed of 4
        7 arpeggio slow  //  Iterate over groups of 4 notes at speed of 8

        If the SFX speed is <= 8, arpeggio speeds are halved to 2, 4


:: Music Editor
```

```
Music in PICO-8 is controlled by a sequence of 'patterns'. Each pattern is a list of
4 numbers indicating which SFX will be played on that channel.


:: Flow control

        Playback flow can be controlled using the 3 buttons at the top right.

        When a pattern has finished playing, the next pattern is played unless:

        - there is no data left to play (music stops)
        - a STOP command is set on that pattern (the third button)
        - a LOOP BACK command is set (the 2nd button), in which case the music player searches
                back for a pattern with the LOOP START command set (the first button) or returns to
                pattern 0 if none is found.

        When a pattern has SFXes with different speeds, the pattern finishes playing when
        the left-most non-looping channel has finished playing. This can be used to set up
        time signatures that don't divide into 32, or double-time drum beats etc.


:: Copying music between or within cartridges

        To select a range of patterns: click once on the first pattern in the pattern
        navigator, then shift-click on the last pattern. Selected patterns can be copied
        and pasted with CTRL-C and CTRL-V. When pasting into another cartridge, the SFX
        that each pattern points to will also be pasted (possibly with a different index)
        if it does not already exist.


:: SFX Instruments

        In addition to the 8 built-in instruments, custom instruments can be defined using
        the first 8 SFX. Use the toggle button to the right of the instruments to select an
        index, which will show up in the instrument channel as green instead of pink.

        When an SFX instrument note is played, it essentially triggers that SFX, but alters
        the note's attributes:

                Pitch is added relative to C2
                Volume is multiplied
                Effects are applied on top of the SFX instrument's effects

        For example, a simple tremelo effect could be implemented by defining an instrument
        in SFX 0 that rapidly alternates between volume 5 and 2. When using this instrument
        to play a note, the volume can further be altered as usual (via the volume channel
        or using the fade in/out effects). In this way, SFX instruments can be used to control
        combinations of detailed changes in volume, pitch and texture.

        SFX instruments are only retriggered when the pitch changes, or the previous note
        has zero volume. This is useful for instruments that change more slowly over time.
        For example: a bell that gradually fades out. To invert this behaviour, effect 3
        (normally 'drop') can be used when triggering the note. All other effect values have
        their usual meaning when triggering SFX instruments.


================================================================================================
        Lua Syntax Primer
================================================================================================

        PICO-8 programs are written using Lua syntax, but do not use the standard Lua library.
        The following is a brief summary of essential Lua syntax.

        For more details, or to find out about proper Lua, see www.lua.org.

        :: Comments

                -- use two hyphens like this to ignore everything until the end of the line
                --[[ multi-line
                comments ]]


        :: Types and assignment

                Types in Lua are numbers, strings, booleans and tables;
```

```
	NUM = 12/100
	S = "THIS IS A STRING"
	B = FALSE
	T = {1,2,3}

	Numbers in PICO-8 are all 16:16 fixed point. They range from -32768.0 to 32767.99

	Hexedecimal notation with optional fractional parts can be used:
		0x11		-- 17
		0x11.4000	-- 17.25

	Dividing by zero evaluates as the largest possible number: 0x7fff.ffff


:: Conditionals

	IF NOT B THEN
		PRINT("B IS FALSE")
	ELSE
		PRINT("B IS NOT FALSE")
	END

	-- with ELSEIF

	IF X == 0 THEN
		PRINT("X IS 0")
	ELSEIF X < 0 THEN
		PRINT("X IS NEGATIVE")
	ELSEIF X > 0 THEN
		PRINT("X IS POSITIVE")
	ELSE
		PRINT("THIS IS LINE IS NEVER REACHED")
	END

	IF (4 == 4) THEN PRINT("EQUAL") END
	IF (4 ~= 3) THEN PRINT("NOT EQUAL") END
	IF (4 <= 4) THEN PRINT("LESS THAN OR EQUAL") END
	IF (4 > 3) THEN PRINT("MORE THAN") END

:: Loops

	FOR X=1,5 DO
		PRINT(X)
	END
	-- prints 1,2,3,4,5

	X = 1
	WHILE(X <= 5) DO
		PRINT(X)
		X = X + 1
	END

	FOR X=1,10,3 DO PRINT(X) END   -- 1,4,7,10

	FOR X=5,1,-2 DO PRINT(X) END  -- 5,3,1


:: Functions and Local Variables

	Y=0
	FUNCTION PLUSONE(X)
		LOCAL Y = X+1
		RETURN Y
	END
	PRINT(PLUSONE(2)) -- 3
	PRINT(Y)          -- 0


:: Tables

	In Lua, tables are a collection of key-value pairs where the key and value types can both
	be mixed. They can be used as arrays by indexing them with integers.

	A={} -- create an empty table
```

```
A[1] = "BLAH"
A[2] = 42
A["FOO"] = {1,2,3}

-- Arrays use 1-based indexing by default

A = {11,12,13,14}
PRINT(A[2]) -- 12

-- The size of a table indexed with sequential 1-based integers:

PRINT(#A)   -- 4

-- Indexes that are strings can be written using dot notation

PLAYER = {}
PLAYER.X = 2 -- is equivalent to PLAYER["X"]
PLAYER.Y = 3

-- see also the tables section in the api reference below.
```

:: PICO-8 Shorthand

PICO-8 also allows several non-standard, shorter ways to write common patterns.

1. IF THEN END statements on a single line can be expressed without the THEN & END

```
IF (NOT B) I=1 J=2

-- is equivalent to: IF (NOT B) THEN I=1 J=2 END
-- everything must be written on the same line, and with the condition in brackets
```

2. unary math operators

```
a += 2  -- equivalent to: a = a + 2
a -= 2  -- equivalent to: a = a - 2
a *= 2  -- equivalent to: a = a * 2
a /= 2  -- equivalent to: a = a / 2
a %= 2  -- equivalent to: a = a % 2
```

3. != operator

Not shorthand, but pico-8 also accepts != instead of ~= for "not equal to"

API

PICO-8 is built on the Lua programming language, but does not include the Lua standard library. Instead, a small api is offered in keeping with PICO-8's minimal design and limited screen space. For an example program that uses most of the api functions, see /DEMOS/API.P8

Functions are written here as:
function_name parameter [optional_parameter]

System functions called from commandline can omit the usual brackets and string quotes:
load blah.p8 --> load("blah.p8")

System

load filename [breadcrumb [param_str]]
save filename

Load or save a cartridge

Returns true iff succeeded

```
        When loading from a running cartridge, the loaded cartridge is immediately run with
        parameter string param_str, and a menu item is inserted with the name breadcrumb, that
        returns the user to the loading cartridge.

        Filenames that start with '#' are taken to be a BBS cart followed by its id:
        load("#1234") -- download [and run] cart number 1234

        If the id is of the cart's parent post, then the latest version is downloaded.
        BBS carts can be loaded from other BBS carts or local carts, but not from exported carts.


folder

        Open the carts folder in the host operating system.

dir (also aliased as ls)

        List files in the current directory. When called from a running program, returns a list
        of all .p8 and .p8.png files in the same directory.

run

        Run from the start of the program
        Can be called from inside a program to reset program.

stop [message]

        Stop the cart and optionally print a message

resume

        Run from the existing cart state (flakey)

reboot

        Reboot the machine
        Useful for starting a new project

info

        Print out some information about the cartridge:
        code size, tokens, compressed size

flip

        Flip the back buffer to screen and wait for next frame (30fps)
        Don't normally need to do this -- _draw() calls it for you.

        If your program does not call flip before a frame is up, and a _draw() callback
        is not in progress, the current contents of the back buffer are copied to screen.

printh str [filename] [overwrite]

        Print a string to host operating system's console for debugging.

        If filename is set, append the string to a file on the host operating system
                // (in the current directory -- use FOLDER to view)
        Setting overwrite to true causes that file to be overwritten rather than appended.
        Use a filename of "@clip" to write to the host's clipboard.
                // use stat(4) to read the clipboard, but the contents of the clipboard are only
                // available after pressing CTRL-V during runtime (for security reasons).

stat x

        Get system status where x is:

        0  Memory useage (0..1024)
        1  CPU used since last flip (1.0 == 100% CPU at 30fps)
        4  Clipboard contents (after user has pressed CTRL-V)
        6  Parameter string
        7  Current framerate

        16..19  Index of currently playing SFX on channels 0..3
        20..23  Note number (0..31) on channel 0..3
        24      Currently playing pattern index
```

```
                25      Total patterns played
                26      Ticks played on current pattern

                80..85  UTC time: year, month, day, hour, minute, second
                90..95  Local time


        extcmd x

                Special system command, only works when running a local cart.
                Where x is a string:

                "label"      set cart label
                "screen"     save a screenshot
                "rec"        set video start point
                "video"      save a .gif to desktop
                "audio_rec"  start recording audio
                "audio_end"  save recorded audio to desktop


-------------------------------------------------------------------------------------------
        Program Structure
-------------------------------------------------------------------------------------------

        There are 3 special functions that, if defined by the user, are called during program
        execution:

                _update()
                        Called once per update at 30fps

                _draw()
                        Called once per visible frame

                _init()
                        Called once on program startup


        _draw() is normally called at 30fps, but if it can not complete in time, PICO-8 will
        attempt to run at 15ps and call _update() twice per visible frame to compensate.


        :: Running PICO-8 at 60fps

                If _update60() is defined instead of _update(), PICO-8 will run in 60fps mode:

                - both _update60() and _draw() are called at 60fps
                - half the PICO-8 CPU is available per frame before dropping down to 30fps

                Note that not all host machines are capable of running at 60fps. Older machines,
                and / or web versions might also request PICO-8 to run at 30 fps (or 15 fps), even
                when the PICO-8 CPU is not over capacity. In this case, multiple _update60 calls
                are made for every _draw call in the same way.


-------------------------------------------------------------------------------------------
        Graphics
-------------------------------------------------------------------------------------------

        PICO-8 has a fixed capacity of 128 8x8 sprites, plus another 128 that overlap with the
        bottom half of the map data ("shared data"). These 256 sprites are collectively called
        the sprite sheet, and can be thought of as a 128x128 pixel image.

        All of PICO-8's drawing operations are subject to the current draw-state. The draw-state
        includes a camera position (for adding an offset to all coordinates), palette mapping
        (for recolouring sprites), clipping rectangle, a drawing colour, and a fill pattern.

        The draw state is reset each time a program is run. This is equivalent to calling:
        clip() camera() pal() color() fillp()

        Colours indexes:

         0  black   1  dark_blue   2  dark_purple   3  dark_green
         4  brown   5  dark_gray   6  light_gray    7  white
         8  red     9  orange     10  yellow       11  green
        12  blue   13  indigo     14  pink         15  peach
```

```
clip [x y w h]

        Sets the screen's clipping region in pixels
        clip() to reset


pget x y
pset x y [c]

        Get or set the colour (c) of a pixel at x, y.


sget x y
sset x y [c]

        Get or set the colour (c) of a spritesheet pixel.


fget n [f]
fset n [f] v

        Get or set the value (v) of a sprite's flag
        f is the flag index 0..7
        v is boolean and can be true or false

        The initial state of flags 0..7 are settable in the sprite editor,
        using the line of little colourful buttons.

        The meaning of sprite flags is up to the user, or can be used to
        indicate which group ('layer') of sprites should be drawn by map.

        If the flag index is omitted, all flags are retrieved/set as a bitfield
        fset(2, 1+2+8)   -- sets bits 0,1 and 3
        fset(2, 4, true) -- sets bit 4
        print(fget(2))   -- 27 (1+2+8+16)


print str [x y [col]]

        Print a string
        If only str is supplied, and the cursor reaches the end of the screen,
        a carriage return and vertical scroll is automatically applied.


cursor x y

        Set the cursor position and carriage return margin


color col

        Set the default color to be used by drawing functions


cls [col]

        Clear the screen and reset the clipping rectangle


camera [x y]

        Set a screen offset of -x, -y for all drawing operations
        camera() to reset


circ     x y r [col]
circfill x y r [col]

        Draw a circle or filled circle at x,y with radius r
        If r is negative, the circle is not drawn

line x0 y0 x1 y1 [col]
```

```
        draw line


rect     x0 y0 x1 y1 [col]
rectfill x0 y0 x1 y1 [col]

        Draw a rectangle or filled rectangle


pal c0 c1 [p]

        Draw all instances of colour c0 as c1 in subsequent draw calls

        pal() to reset to system defaults (including transparency values and fill pattern)
        Two types of palette (p; defaults to 0)
                0 draw palette   : colours are remapped on draw    // e.g. to re-colour sprites
                1 screen palette : colours are remapped on display // e.g. for fades
        c0 colour index 0..15
        c1 colour index to map to


palt c t

        Set transparency for colour index to t (boolean)
        Transparency is observed by spr(), sspr() and map()
        e.g. palt(8, true) -- red pixels not drawn
        palt() resets to default: all colours opaque except colour 0



spr n x y [w h] [flip_x] [flip_y]

        draw sprite n (0..255) at position x,y
        width and height are 1,1 by default and specify how many sprites wide to blit.
        Colour 0 drawn as transparent by default (see palt())
        flip_x=true to flip horizontally
        flip_y=true to flip vertically


sspr sx sy sw sh dx dy [dw dh] [flip_x] [flip_y]

        Stretch rectangle from sprite sheet (sx, sy, sw, sh) // given in pixels
        and draw in rectangle (dx, dy, dw, dh)
        Colour 0 drawn as transparent by default (see palt())
        dw, dh defaults to sw, sh
        flip_x=true to flip horizontally
        flip_y=true to flip vertically

fillp p

        The PICO-8 fill pattern is a 4x4 2-colour tiled pattern observed by:
                circ() circfill() rect() rectfill() pset() line()

        p is a bitfield in reading order starting from the highest bit. To calculate the value
        of p for a desired pattern, add the bit values together:

                .-----------------------.
                |32768|16384| 8192| 4096|
                |-----|-----|-----|-----|
                | 2048| 1024| 512 | 256 |
                |-----|-----|-----|-----|
                | 128 |  64 |  32 |  16 |
                |-----|-----|-----|-----|
                |  8  |  4  |  2  |  1  |
                '-----------------------'

        For example, FILLP(4+8+64+128+  256+512+4096+8192) would create a checkerboard pattern.

        This can be more neatly expressed in binary: FILLP(0b0011001111001100)
        The default fill pattern is 0, which means a single solid colour is drawn.

        To specify a second colour for the pattern, use the high bits of any colour parameter:

                FILLP(0b0011010101101000)
                CIRCFILL(64,64,20, 0x4E) -- brown and pink
```

```
An additional bit 0b0.1 can be set to indicate that the second colour is not drawn.

        FILLP(0b0011010101101000.1) -- checkboard with transparent squares
```

Tables

---

add t v

```
Add value v to the end of table t.
Equivalent to t[#t+1] = v

        FOO={}        -- create empty table
        ADD(FOO, 11)
        ADD(FOO, 22)
        PRINT(FOO[2]) -- 22
```

del t v

```
Delete the first instance of value v in table t
The remaining entries are shifted left one index to avoid holes.
Note that v is the value of the item to be deleted, not the index into the table!
del() can be called safely on a table's item while iterating over that table.

        A={1,10,2,11,3,12}
        FOR ITEM IN ALL(A) DO
                IF (ITEM < 10) THEN DEL(A, ITEM) END
        END
        FOREACH(A, PRINT) -- 10,11,12
        PRINT(A[3])       -- 12
```

all t

```
Used in FOR loops to iterate over all items in a table (that have a 1-based integer index),
in the order they were added.

        T = {11,12,13};
        ADD(T,14)
        ADD(T,"HI")
        FOR V IN ALL(T) DO PRINT(V) END -- 11 12 13 14 HI
        PRINT(#T) -- 5
```

foreach t f

```
For each item in table t, call function f with the item as a single parameter.

        FOREACH(T, PRINT)
```

pairs t

```
Used in FOR loops to iterate over table t, providing both the key and value for each item.
Unlike all(), pairs() iterates over every item regardless of indexing scheme.
Order is not guaranteed.

        T = {["HELLO"]=3, [10]="BLAH"}
        T.BLUE = 5;
        FOR K,V IN PAIRS(T) DO
                PRINT("K: "..K.."  V:"..V)
        END

Output:

        K: 10  v:BLAH
        K: HELLO  v:3
        K: BLUE  v:5
```

Input

```
-------------------------------------------------------------------------------------------

        btn [i [p]]

                get button i state for player p (default 0)
                i: 0..5: left right up down button_o button_x
                p: player index 0..7

                If no parameters supplied, returns a bitfield of all 12 button states for player 0 & 1
                        // P0: bits 0..5  P1: bits 8..13

                Default keyboard mappings to player buttons:
                        player 0: cursors, Z,X / C,V / N,M
                        player 1: ESDF, LSHIFT,A / TAB,Q,E


        btnp [i [p]]

                btnp is short for "Button Pressed"; Instead of being true when a button is held down,
                btnp returns true when a button is down AND it was not down the last frame. It also
                repeats after 15 frames, returning true every 4 frames after that (at 30fps -- double
                that at 60fp). This can be used for things like menu navigation or grid-wise player
                movement.



-------------------------------------------------------------------------------------------
        Audio
-------------------------------------------------------------------------------------------

        sfx n [channel [offset [length]]]

                play sfx n on channel (0..3) from note offset (0..31) for length notes
                n -1 to stop sound on that channel
                n -2 to release sound on that channel from looping
                Any music playing on the channel will be halted
                offset in number of notes (0..31)

                channel -1 (default) to automatically choose a channel that is not being used
                channel -2 to stop the sound from playing on any channel


        music [n [fade_len [channel_mask]]]

                play music starting from pattern n (0..63)
                n -1 to stop music
                fade_len in ms (default: 0)
                channel_mask specifies which channels to reserve for music only
                        e.g. to play on channels 0..2: 1+2+4 = 7

                Reserved channels can still be used to play sound effects on, but only when that
                channel index is explicitly requested by sfx().


-----------------------------------------------------------------------------------------------------
        Map
-----------------------------------------------------------------------------------------------------

        mget x y
        mset x y v

                get or set map value (v) at x,y

        map cel_x cel_y sx sy cel_w cel_h [layer]

                draw section of map (in cels) at screen position sx, sy (pixels)
                if layer is specified, only cels with the same flag number set are drawn
                        // Bitfield. So 0x5 means draw sprites with bit 0 and bit 2 set.
                        // defaults to all sprites

                exception: sprite 0 is always taken to mean empty.

                e.g. map(0,0, 20,20, 4,2)
                -> draws a 4x2 blocks of cels starting from 0,0 in the map, to the screen at 20,20
```

```
-------------------------------------------------------------------------------------------------------------
        Memory
-------------------------------------------------------------------------------------------------------------

        PICO-8 has 3 types of memory:

                1. Base ram (32k): see layout below. Access with peek() poke() memcpy() memset()
                2. Cart rom (32k): same layout as base ram until 0x4300. Copy from cart to base ram with reload()
                3. Lua ram (2MB): compiled program + variables. Pay no attention to the man behind the curtain.

                Technical note: // you probably don't need to know this
                While using the editor, the data being modified is in cart rom, but api functions such as spr()
                and sfx() only operate on base ram. PICO-8 automatically copies cart rom to base ram (i.e.
                calls reload()) in 3 cases:

                        1. When a cartridge is loaded
                        2. When a cartridge is run
                        3. When exiting any of the editor modes


        :: Base ram memory layout
                0x0    gfx
                0x1000 gfx2/map2 (shared)
                0x2000 map
                0x3000 gfx_props
                0x3100 song
                0x3200 sfx
                0x4300 user data
                0x5e00 persistent cart data (256 bytes)
                0x5f00 draw state
                0x5f40 hardware state
                0x5f80 gpio pins (128 bytes)
                0x6000 screen (8k)

                User data has no particular meaning and can be used for anything via memcpy(), peek() & poke().
                Persistent cart data is mapped to 0x5e00..0x5eff but only stored if cartdata() has been called.
                Colour format (gfx/screen) is 2 pixels per byte: low bits encode the left pixel of each pair.
                Map format is one byte per cel, where each byte normally encodes a sprite index.


        peek addr
        poke addr val

                Read and write one byte to an address in base ram.
                Legal addresses are 0x0..0x7fff
                Reading or writing outside causes a fault

        memcpy dest_addr source_addr len

                Copy len bytes of base ram from source to dest
                Sections can be overlapping

        reload dest_addr source_addr len [filename]

                Same as memcpy, but copies from cart rom
                The code section ( >= 0x4300) is protected and can not be read.
                If filename (.p8) specified, load data from a different cartridge.

        cstore dest_addr source_addr len [filename]

                Same as memcpy, but copies from base ram to cart rom
                cstore() is equivalent to cstore(0, 0, 0x4300)
                Can use for writing tools to construct carts or to visualize the state
                of the map / spritesheet using the map editor / gfx editor.
                The code section ( >= 0x4300) is protected and can not be written to.

                If a filename is specified, the data is written directly to that
                cartridge on disk. Up to 64 cartridges can be written in one session.
                Currently only the .p8 format is supported.
                See the 'Cartridge Data' section for additional notes on using cstore.


        memset dest_addr val len

                Set len bytes to val
```

```
                (quite fast -- can use to draw unclipped horizonal scanlines etc)


-------------------------------------------------------------------------------------------------------------
        Math
-------------------------------------------------------------------------------------------------------------

        max x y
        min x y
        mid x y z

                Returns the maximum, minimum, or middle value of parameters
                For example, mid(7,5,10) returns 7

        flr x

                Returns the closest integer below x // x-(x%1)
                flr(4.1)  --> 4
                flr(-2.3) --> -3.0


        cos x
        sin x

                Returns the cosine of x, where 1.0 indicates a full circle
                sin is inverted to suit screenspace
                e.g. sin(0.25) returns -1

                If you'd prefer radian-based trig functions without the y inversion,
                paste the following snippet near the start of your program:

                cos1 = cos function cos(angle) return cos1(angle/(3.1415*2)) end
                sin1 = sin function sin(angle) return sin1(-angle/(3.1415*2)) end


        atan2 dx dy

                Converts dx, dy into an angle from 0..1
                As with cos/sin, angle is taken to run anticlockwise in screenspace
                e.g. atan(1, -1) returns 0.125

        sqrt x

                Return the square root of x

        abs x

                Returns the absolute (positive) value of x

        rnd x
                Returns a random number n, where 0 <= n < x
                If you want an integer, use flr(rnd(x))

        srand x
                Sets the random number seed
                The seed is automatically randomized on cart startup

        Bitwise operations

                band x y
                bor  x y
                bxor x y
                bnot x

                shl  x y
                shr  x y

                // shifts are logicial shifts (the sign bit is not shifted)

-------------------------------------------------------------------------------------------------------------
        Custom Menu Items
-------------------------------------------------------------------------------------------------------------

        menuitem index [label callback]
```

```
                Add an extra item to the pause menu

                Index should be 1..5 and determines the order each menu item is displayed
                label should be a string up to 16 characters long
                callback is a function called when the item is selected by the users

                When no label or function is supplied, the menu item is removed

                example:
                menuitem(1, "restart puzzle", function() reset_puzzle() sfx(10) end)


-------------------------------------------------------------------------------------------------------
        Strings
-------------------------------------------------------------------------------------------------------

        s = "the quick brown fox"

        -- length

                print(#s)          --> 19


        -- joining strings

                print("three "..4) --> "three 4"


        -- sub() to grab substrings

                print(sub(s,5,9))  --> "quick"
                print(sub(s,5))  --> "quick brown fox"


-------------------------------------------------------------------------------------------------------
        Types
-------------------------------------------------------------------------------------------------------

        type val

                returns the name of the type of value x as a string

        tostr val [hex]

                returns val as a string

                iff hex is true and val is a number, an unsigned hexidecimal writing of
                the number is returned in the format "0x0000.0000". You can use this to
                inspect the internal representation of PICO-8 numbers.

                iff val is a boolean, it is written as "true" or "false"

                all other val types are written as "[typename]"

        tonum val

                converts val to a number
                iff val is a string, the number is taken to be decimal unless prefixed with "0x"
                if the conversion fails, tonum returns nil.

-----------------------------------------------------------------------------------------
        Cartridge Data
-----------------------------------------------------------------------------------------

        Each cartidge is able to store 64 numbers (256 bytes) of persistent data
        on the user's PICO-8 (rather than on the cart itself). This can be used as
        a lightweight way to store things like high scores or to save player progress.

        If you need more than 256 bytes, it is also possible to write directly to the
        cartridge using cstore(). The disadvantage is that the data is tied to that
        particular version of the cartridge. e.g. if a game is updated, players will
        lose their savegames.

        Another alternative is to write directly to a second cartridge by specifying
        a fourth parameter to cstore(). This requires a cart swap though (so is
```

```
slightly slower), and leaves data-cart litter when run from a local folder.
The file should be in .p8 format: cstore(0,0,0x2000,"spritesheet.p8")


cartdata id

        Call this once on cartridge load. id is a string up to 64 characters long, and
        should be unusual enough that other cartridges do not accidentally use the
        same id.

        e.g. cartdata("zep_jelpi")

        legal characters are a..z, 0..9 and underscore (_)

        returns true iff data was loaded.

        cartdata can not be called more than once per cartridge execution.

        Once a cartdata id has been set, the area of memory 0x5e00..0x5eff is mapped
        to permanent storage, and can either be accessed directly or via dget/dset.


dget index

        Get the number stored at index (0..63)
        Use this only after you have called cartdata()

dset index value

        Set the number stored at index (0..63)
        Use this only after you have called cartdata()

        There is no need to flush written data -- it is automatically
        saved to permanent storage even if POKE()'ed directly.


-------------------------------------------------------------------------------------------
        GPIO
-------------------------------------------------------------------------------------------

        GPIO stands for "General Purpose Input Output", and allows machines to comunicate with
        each other. PICO-8 maps bytes in the range 0x5f80..0x5fff to gpio pins that can be
        poke()ed (to output a value -- e.g. to make an LED light up) or peek()ed (e.g. to read
        the state of a switch).

        GPIO means different things for different host platforms:

        CHIP:         0x5f80..0x5f87 mapped to xio-p0..xio-p7
        Pocket CHIP:  0x5f82..0x5f87 mapped to GPIO1..GPIO6
                // xio-p0 & p1 are exposed inside the prototyping area inside the case.
        Raspberry Pi: 0x5f80..0x5f8f mapped to wiringPi pins 0..7
                // see http://wiringpi.com/pins/ for mappings on different models

        CHIP and Raspberry Pi values are all digital: 0 (LOW) and 255 (HIGH)

        Note that to have access to gpio pins, you may need to run pico8 as root:

        sudo pico8

        A simple program to blink any LEDs attached on and off:

        t = 0
        function _draw()
         cls(5)
         for i=0,7 do
          val = 0
          if (t % 2 < 1) val = 255
          poke(0x5f80 + i, val)
          circfill(20+i*12,64,4,val/11)
         end
         t += 0.1
        end

        :: HTML
```

Cartridges exported as HTML / .js use a global array of integers (pico8_gpio) to represent gpio pins. The shell HTML should define the array:

```
var pico8_gpio = Array(128);
```

Additional Lua Features

PICO-8 also exposes 2 features of Lua for advanced users: Metatables and Coroutines.

For more information, please refer to the Lua 5.2 manual.

:: Metatables

Metatables can be used to define the behaviour of objects under particular operations. For example, to use tables to represent 2D vectors that can be added together, the '+' operator is redefined by defining an "__add" function for the metatable:

```
vec2d={
 __add=function(a,b)
        return {x=(a.x+b.x), y=(a.y+b.y)}
 end
}

v1={x=2,y=9} setmetatable(v1, vec2d)
v2={x=1,y=5} setmetatable(v2, vec2d)
v3 = v1+v2
print(v3.x..","..v3.y) -- 3,14
```

`setmetatable t, m`

set table t metatable to m

`getmetatable t`

return the current metatable for table t, or nil if none is set

:: Coroutines

Coroutines offer a way to run different parts of a program in a somewhat concurrent way, similar to threads. A function can be called as a coroutine, suspended with yield() any number of times, and then resumed again at the same point.

```
function hey()
        print("doing something")
        yield()
        print("doing the next thing")
        yield()
        print("finished")
end

c = cocreate(hey)
for i=1,3 do coresume(c) end
```

`cocreate f`

Create a coroutine for function f.

`coresume c [p0 p1 ..]`

Run or continue the coroutine c. Parameters p0, p1.. are passed to the coroutine's function.

Returns true iff the coroutine completes without any errors
Returns false, error_message if there is an error.

** Runtime errors that occur inside coroutines do not cause the program to stop running. It is a good idea to wrap coresume() inside an assert(). If the assert fails, it will print the error message generated by coresume.

```
                        assert(coresume(c))


                costatus c

                        Return the status of coroutine c as a string:
                                "running"
                                "suspended"
                                "dead"

                yield

                        Suspend execution and return to the caller.


-------------------------------------------------------------------------------------------
        VERSION HISTORY
-------------------------------------------------------------------------------------------

        v0.1.11c

                Added: Local and UT time queries using stat()
                Added: host_framerate_control (config.txt) to improve performance on slower machines and web
                Added: Control over cpu useage when running in background (-background_sleep_ms / config.txt)
                Added: Windows icon in exported exe
                Added: F11 to toggle fullscreen
                Added: export -c switch to indicate transparent icon colour
                Added: show_backup_messages (config.txt) to turn off backup notifications
                Added: SFX instruments documentation in pico8.txt
                Added: Error message when trying to export carts with code size over the compressed limit
                Changed: If config.txt is not found, the same directory as the executable is searched
                Changed: If sdl_controllers.txt exists in the same directory as the executeable, it is processed first
                Changed: Shorthand if () statements must be written on a single line
                Fixed: reload() from bundled, non-primary cart in exported html multicart reads only original data
                Fixed: Exported binaries wrongly observe F7 (capture label)
                Fixed: Loading carts from earlier versions alters SFX data not intended for audio
                Fixed: Old version of fill patterns documentation near end of pico8.txt
                Fixed: 'backed up unsaved changes' message displayed during runtime for cstored() carts
                Fixed: PICO-8 runs too slowly when in background (new default background_sleep_ms: 20)
                Fixed: Saving screenshots and videos from exported binaries are named 0_*
                Fixed: Compressed size limit warning on save doesn't mention exported carts
                Fixed: btn(), btnp() don't work in infinite loops
                Fixed: btnp() timing inconsistent between 30fps / 60fps / during frame-skipping / with no _update
                Fixed: Can't move between channels while music is playing in song mode


        v0.1.11b

                Fixed: Preprocessor bug regressions: "if (..) or", "a.b -= c - d"
                Fixed: Crash when pressing menu button on an empty favourites list


        v0.1.11

                Added: Binary exporters (Windows, Linux, Mac OSX)
                Added: Code tabs
                Added: Splore cart menu
                Added: Fill patterns
                Added: Custom sfx instruments
                Added: load("#1234") to load [and run] a BBS cart
                Added: -x switch // execute a cart headless, for making command-line tools
                Added: Compressed size display and limit warning lights in code editor
                Added: CTRL-L to jump to a line number in code editor
                Added: numbers can be written in binary: 0b10100010
                Added: tostr(), tonum()
                Added: extcmd(): audio_rec, audio_end to record all audio output.
                Added: ls() returns a list of local files if called while running
                Added: getmetatable()
                Added: coroutine error reporting // wrap coresume() in assert()
                Added: sfx() can take a 4th parameter: number of notes to play
                Added: Live sfx and music editing + better navigation controls
                Added: Transpose selected sfx notes relative to C by entering a note w/ SHIFT held
                Added: Insert and delete sfx rows with enter and backspace
                Added: Hidden note data is shown in sfx editor when relevant (slide, arps)
```

```
        Added: Warning displayed when unsaved changes backed up
        Added: Separate animation for downloading vs. loading a cart
        Added: export -p switch to supply a customized html template
        Added: Mousewheel when devkit mouse enabled: stat(36) // not supported in web
        Added: < > to change zoom level in gfx and map editors
        Changed: Rebalanced / fixed api cpu costs
        Changed: Screenshot and gif filenames based on current cart if available
        Changed: add() returns the added object
        Changed: removed global hpf on audio
        Changed: (sfx) can slide to volume 0
        Changed: removed master low pass filter
        Changed: assert() can take an optional error_message parameter
        Changed: ? (shorthand for print()) can be prefixed by whitespace
        Changed: shl(), shr() return 0 if second parameter >= 32
        Changed: Automatically drop down to software blitting mode if opengl fails
        Changed: Lua memory limit set to 2MB (was 1MB)
        Changed: Some options (-width, -show_fps) apply only to the session; not saved to config.txt
        Updated: Internal game controller mappings from SDL_GameControllerDB
        Fixed: Pops & clicks in audio when switching between playing SFX
        Fixed: Crash in audio mixer because of bad locking
        Fixed: Crash when loading .p8 files with more than 64k of code
        Fixed: Indexing of sparse tables fails after removing n/2 elements
        Fixed: Calling stat() inside an infinite loop crashes
        Fixed: Resetting cartridge corrupts cartridge data in range 0x5e00..0x5eff
        Fixed: Can not recover from a cart error caused by glitchy data on resetting
        Fixed: String->negative number conversion off by 0x0.0001 (-1 --> 0xffff0001)
        Fixed: Crash when running cart closed to 64k char limit
        Fixed: Cursor can't move to the right of last character in code editor
        Fixed: Missing highlighted keywords: in, add, del, menuitem
        Fixed: Preprocessor bugs: "a+=1+2\n*3", "a+=(1)ba=42", "a[(1)]+=1"
        Fixed: Preprocessor performs replacements inside a string printed with ?
        Fixed: Display freezes when terminating a program running at >100% cpu
        Fixed: Quick-running (CTRL-R) clobbers some editor state (e.g. current sprite page)
        Fixed: Loading a .p8 file with a future version reports a generic failure
        Fixed: alt-enter to toggle fullscreen also triggers pause menu
        Fixed: Splore scrolling jumps around when list gets too long


v0.1.10c

        Fixed: atan flips sign for very negative values of x close to zero


v0.1.10b
        Fixed: HTML exporter carts don't run
        Fixed: HTML export 60fps support broken
        Fixed: HTML export when path has a space in it (common for OSX)
        Fixed: atan2 ignores sign of y
        Fixed: (Raspberry Pi) Crash when access gpio not as root


v0.1.10

        Added: Multi-cart export in html
        Added: Cart reset glitch
        Added: Demo carts: bounce, sort
        Added: .p8 format can now store cart labels
        Added: Splore navigation keys: pageup/down, home, end
        Added: Splore useage hint shown on empty favourites list
        Added: Warning on boot when data folder is read-only or can't be created
        Added: Pressing tab with code selected indents those lines (shift-tab to un-indent)
        Added: Double click word to select it
        Added: Trigger screenshot/video/label capture from inside program: extcmd()
        Changed: CTRL+left/right in code editor skips to end of word or span of non-whitespace
        Changed: When a cart terminates from splore, button press is required to continue
        Changed: load("@clip") can only be called from commandline (security)
        Fixed: Can over-allocate host memory if exceed it within one frame
        Fixed: atan2(-1, -32768) crash, and error for small values of dy
        Fixed: (Web) using cstore() on self causes unloadable cart (bug introduced in 0.1.8?)
        Fixed: (web) Pressing ctrl-v crashes the player (should do nothing)
        Fixed: (Raspberry Pi) WiringPi library required in static build
        Fixed: (Raspberry Pi) Crash on exit when launching via desktop icon
        Fixed: (Raspberry Pi) keyboard input broken (observed on raspi2s)
```

v0.1.9b

```
Added: Alternative function key mapping: ctrl-6..9 for F6..F9
Added: Alternative glyph entry method: (ctrl-k) to toggle glyph mode
Changed: Enter glyphs with shift a..z, but can be disabled in config.txt
Changed: Increased emscripten ram to 128MB (some carts at risk of running out)
Fixed: Crash when window size is tiny or minified
Fixed: Crash on toggling fullscreen mode
Fixed: printh can write files outside filetree (security issue)
Fixed: show_fps (can also now be toggled with ctrl-1)
Fixed: Shorthand if/then syntax error when using the form: (functionname)(param)
Fixed: log.txt not saved in path specified by -home switch
Fixed: Default application data folder created even when -home specified
Fixed: Missing dynamic builds (pico8_dyn) from linux archives
Fixed: Removed unneeded RPATH from linux binaries
Fixed: export foo%d.wav fails to write multiple files
```

v0.1.9

```
Added: Copy and paste sprites and whole cartridges directly to BBS posts
Added: JAM category in splore
Added: GPIO support for Raspberry Pi
Added: Read clipboard using stat(4) after user presses CTRL-V
Added: printh() can optionally write to a file or the host clipboard
Added: Editor tool information and tips shown on mouseover
Added: Set desktop path with -desktop (screenshots and gifs are saved here)
Added: Warning on saving .p8 when compressed code size exceeds .p8.png limit
Added: Alternative editor colours // config.txt: gui_theme 1
Added: Dotted line every 8 rows in song view
Added: -screenshot_scale (default: 3) and -gif_scale (default: 2)
Added: Can use ctrl-up, ctrl-down to jump to start and end of code
Added: CTRL-M to mute/unmute sound
Added: HTML5-exported carts support 60fps
Added: Timeout switch for splore downloads: -timeout
Changed: Glyph characters typed with alt + a..z
Changed: stat(0) does not include allocations waiting to be garbage collected
Changed: Unfiltered screen stretching at integer scales by default
Changed: Removed -aspect and -scale settings (use draw_rect instead)
Fixed: -home has no effect under Windows
Fixed: Sometimes frame skipping starts before CPU useage has reached 100%
Fixed: Double-speed BTNP() timing in 60fps mode
Fixed: Exported HTML fails when _update60 is used instead of _update
Fixed: Can't copy and paste button glyphs
Fixed: Lines containing glyphs do not scroll far enough horizontally
Fixed: Loading .p8 renamed as .p8.png from splore freezes
Fixed: Bucketfill in map doesn't sync to shared memory
Fixed: fset fails when de-setting flags
Fixed: Syntax error when beginning with the form: IF (..) [OR|AND]\n
Fixed: cls() costs twice as much cpu as it should
Fixed: wav file exporter missing some data / writing truncated buffers
Fixed: Entering new notes in song view doesn't observe current volume, instrument
Fixed: alt-tab sometimes generates alt character text entry event
Fixed: Resuming a cancelled download in splore causes crash
Fixed: Controller attributes in log.txt always shown as -1
```

v0.1.8

```
Added: 60fps support
Added: Music exporter
Added: Custom GIF length (maximum 120 seconds)
Added: -,+ to navigate sprite tabs, sfx, music patterns
Added: sfx editor: navigate with home, end, pageup/down, mousewheel
Added: <, > to modify sfx speed, or click and drag
Added: Middle mouse button to pan around spritesheet / map
Added: Shortcut command for splore: S
Added: Pre-installed selection of BBS cart (use INSTALL_GAMES)
Added: Warning when saving .p8.png with no label
Added: (OSX) logging to ~/Library/Logs (viewable with Console.app)
Added: -pixel_perfect switch (on by default)
Added: -draw_rect switch
Changed: Can not CTRL-S save over a loaded bbs cart
Changed: Only .p8 files listed by dir() and by splore
Changed: Command history increased to 256
Changed: exit() / shutdown() have no effect while running cart
```

```
Fixed: Memory useage (stat(0)) inconsistent across host platforms
Fixed: Spinny disks shows when reloading current cart with load()
Fixed: GIF saver does not respect 64x64 / mirrored modes
Fixed: Miscellaneous multi-line comments / strings issues
Fixed: Empty map cels cost cpu in mapdraw()
Fixed: mapdraw() slowdown when drawing bottom half of map
Fixed: preprocess changes semantics when += and : operators on same line
Fixed: Identifiers starting with underscore counted as extra token
Fixed: Saving .png exceeding compressed code limit fails silently
Fixed: Right-clicking a sprite does not set the currently edited sprite
Fixed: (Windows) extra space added to pasted lines
Fixed: spr() expensive when drawn with low negative coordinates
Fixed: pipe character identical to colon character
Fixed: (Raspberry Pi) shift key appends a character when entering text
Fixed: Editor mode buttons are still clickable during cart runtime
Fixed: When loading a .p8.png file, label is reset and needs to be re-captured
Fixed: export() does not report failure
Fixed: mset()'d changes in shared memory not readable via peek() / sget()
Fixed: cstore() saving edited code
Fixed: audio pop between patterns during music playback
```

v0.1.7

```
Added: menuitem()
Added: button glyphs in code (shift-L, R, U, D, X, O)
Added: Customisable data directory (e.g. pico8 -home mydata)
Added: Web gpio pins: read and write pico8_gpio[] in javscript
Fixed: SPLORE search doesn't reset
Fixed: Splore skipping 33rd cart listing after loading more items
Fixed: Crash when selecting a local binary file in splore
Fixed: Semicolon can't be used as a list or statement separator
Fixed: Exported html can not cstore self
```

v0.1.6

```
Added: SPLORE local & bbs cartridge explorer
Added: setmetatable(), cocreate(), coresume(), costatus(), yield()
Added: Spinning cart icon to show when a cart is swapped / written to
Added: Permanent storage when carts played in a browser
Added: Adjustable aspect ratio (-aspect 420 for 1:1)
Changed: Lua memory limit: 1024k (was 512k)
Changed: Music channel now resumes after being clobbered by an sfx
Changed: Arpeggios double speed when SFX speed <= 8
Changed: Exceeding compressed code limit does not block saving in .p8 format
Changed: spr() half as expensive, to be consistent with map()
Changed: Fractional hex number notation: 0x0.3 == 0x0.3000, (was 0x0.0003)
Changed: : operator doesn't count as an extra token (same as .)
Changed: cstore() writes directly to disk
Changed: cstore(), reload() return number of bytes read / written
Changed: save() while running does nothing. (use cstore() instead)
Changed: load() while running loads and runs the specified cartridge
Fixed: Small pops in audio mixer caused by sound wave discontinuities
Fixed: HTML5-exported sound clicks badly under Chrome
Fixed: Display palette is not oberserved when exporting GIFs
Fixed: Rapid keypresses causes duplicate readings in tracker & text editor
Fixed: += inside comments breaks preprocessor
Fixed: sspr() cpu cost the same when clipped
Fixed: cartdata() with bad parameters crashes
Fixed: EXPORT from commandline can not be used without brackets and quotes
```

v0.1.5

```
Added: Raspberry Pi Build
Added: Keyboard configuration for player buttons (KEYCONFIG)
Added: Music tracker select / copy / paste
Added: Single-level undo in audio tools
Added: Live preview of frequencies in sound editor
Fixed: Command history extends past last reboot
Fixed: Sfx exporter broken
Fixed: Slashes at end of path resolve to double slashes
Fixed: Load cart from commandline under Windows
```

```
v0.1.4d
v0.1.4c

        Fixed: International character entry inserting extra characters
        Fixed: Lines with tabs have broken cursor placement and display boundary

v0.1.4b

        Fixed: OSX command-key combinations broken

v0.1.4

        Added: spritesheet importing and exporting with import("blah.png"), export("blah.png")
        Added: sfx exporting with export("blah%d.wav")
        Added: External cartridge parameter for reload() and cstore()
        Added: Persistent cartridge data mapped to 0x5e00
        Added: Click token limit to toggle token & char limit display
        Added: assert(), type()
        Added: P to pause
        Changed: code char limit: 64k (was 32k)
        Changed: local declarations and semicolons not counted as tokens
        Changed: Pairs of brackets and block delimitations count as one token
        Changed: Only _update() or _draw() need to exist to enter main loop
        Changed: Allow forward-slash in code editor
        Changed: info() reports current (last loaded or saved) filename
        Changed: html5 version compiled with NO_DYNAMIC_EXECUTION
        Changed: can only cstore up to 64 different files in one session
        Changed: load() automatically copies data section of cart to base ram
        Fixed: Shift-drag-copy sprites -> paste only pastes 1x1
        Fixed: ".." should count as one token
        Fixed: Tracker displaying D instead of .
        Fixed: Multi-line comments
        Fixed: Crash on run when code close to char limit
        Fixed: When over token limit, can not run any command
        Fixed: Unused high bits in SFX section not saved in .p8 format
        Fixed: Camera position memory mapping out of sync
        Fixed: pico8.txt link broken in windows installer
        Fixed: print() crashes when parameter is not a string or numbers
        Fixed: Multi-line strings & escape chars mess up tokenizer and print()
        Fixed: Joystick not responding when left stick is up to the left
        Fixed: Alt-F4 saves screenshot before quitting
        Fixed: Sprite editor mode button doesn't show fullscreen mode
        Fixed: -sound parameter not working in html5 version


v0.1.3

        Added: paste into commandline
        Fixed: lua standard libraries accessible
        Fixed: command-line loading doesn't work
        Fixed: music pattern finished too early when all tracks set to looping
        Fixed: peek()ing odd bytes in sfx address space masks bit 7
        Fixed: cstore and reload from code space should have no effect

v0.1.2

        Added: html5 cartridge exporter
        Added: Cartridge save data (64 fixed point numbers)
        Added: 8-player input
        Added: Demo carts: COLLIDE and BUTTERFLY
        Added: Command-line parameters // load cart, -run, settings
        Added: Alternative function keys (F6..F9 aliased as F1..F4)
        Added: pairs()
        Added: printh() for debugging
        Added: Tab completion for filenames in console
        Added: stack trace on runtime error
        Changed: music pattern length taken to be first non-looping channel's length
        Changed: noise instrument (6) has low frequency white noise scaled by volume
        Changed: screenshot captures whole window contents at display resolution
        Changed: del() moves remaining items up one index to maintain a packed table
        Changed: add(),del(),count(),all() no longer store extra fields
        Changed: removed count() from docs -- now just a legacy function. Use # operator instead.
        Changed: cursor only blinks while window is active
        Changed: peek(), poke() and binary operations (band()..) have no function call overhead
        Changed: yellow slightly warmer
```

```
Changed: No camera snapping after pan in map mode
Fixed: sqrt() crashing for 0 or >= 32761
Fixed: Semi-colon characters in text editor
Fixed: Long lines split when saving in .p8 format
Fixed: pget() does not respect camera position
Fixed: Error message when peeking or poking outside of legal address space
Fixed: Search replace colour fills one pixel outside of selected region
Fixed: Playing an empty music pattern breaks subsequent music playback
Fixed: Invalid sfx editing state on startup
Fixed: Painting instruments values in frequency view also sets volumes
Fixed: Inconsistent gif recording speeds
Fixed: Unmapped joystick support
Fixed: Compressed code size sometimes larger than uncompressed
Fixed: mid() fails when first argument is not smallest
Fixed: Scroll wheel changes sprite/map zoom while in code editor
Fixed: CTRL-R (quick-run) drawing over current line in command mode
Fixed: Label capture (F7) does not respect screen palette state
Fixed: Syntax highlighting of api functions and hex numbers
Fixed: Looping to 0 with negative step finishes at 1
Fixed: nil values printed as false instead of nil
Fixed: Hexedecimal fractional parts
Fixed: btnp() unresponsive when skipping frames
Fixed: Editing mode is lost when using ctrl-r to run
Fixed: Tracker note entry keys mapped, messing up piano-like layout
Fixed: Shared gfx/map memory out of sync after some editor operations
Fixed: Alt-gr character entry
Fixed: Can map display palette to entries >= 16 using poke()
Fixed: Using shift to select in code editor has wrong selection range
Fixed: Dragging above top of text causes selection to flip to end
Fixed: Duplicate at end of file listing
```

v0.1.1

```
Added: Token-based code limiting (8192 tokens, 32k ascii text)
Added: Freeform move, pan and selection in sprite and map editors
Added: Flood-fill tool (sprite and map)
Added: .GIF saver
Added: CTRL-Stamp to stamp with transparency
Added: Single-step undo for map and sprites
Added: 2x2 brush
Added: sqrt(), atan2()
Added: CTRL-S to quick-save
Added: CTRL-R reloads .p8 file and runs (useful for external text editing)
Added: Automatic backups on overwriting or quitting without saving
Added: Scroll wheel zooms in sprite editor
Added: Customisable resolution //  e.g. pico8 -width 580
Added: Strings highlighted as green
Added: ALT-click can optionally simulate right click (see config.txt)
Added: palt() to control transparency for spr(), sspr()
Added: info()
Changed: load() tries adding .p8.png, .png if file doesn't exist
Changed: Draw operations apply only to selection when active
Changed: Move operations (cursors) apply to selection if present
Changed: Removed time()
Changed: Random seed is random on cart startup
Changed: api functions never read directly from cart rom
Changed: sspr() can take negative values for dw, dh
Fixed: Sparse table indexing with integers fails
Fixed: Assignment operators and shortform if-then-else failing
Fixed: sspr() failed when w0 == 128
Fixed: Circle drawing broken when camera not (0,0)
Fixed: CPU hogging
Fixed: Noise instrument clobbers rnd() sequence
Fixed: Audio system not resetting on program reset
Fixed: % operator sometimes wrong for negative values
Fixed: Length operator (#)
Fixed: Power operator (^)
Fixed: Line clipping bug on right and bottom edges
Fixed: print() precision for whole numbers
Fixed: print() broken for negative y values
Fixed: tokenization and keyword highlighting
Fixed: sprite properties not copied/pasted
Fixed: Only sfx 0..32 could be used as music patterns
Fixed: Saving and loading a .p8 file adds newline to end of code
```

```
        Fixed: Drag selection to left margin in code editor -> selects all


v0.1.0

        Added: demo cart: hello.p8 (use install_demos)
        Added: CTRL-R from anywhere to run cart or restart cart
        Added: use a,s to select colour in gfx editor
        Added: consistent operation cpu costs
        Added: btn(), btnp() with no arguments returns bitfield
        Added: fget(id) returns bitfield of that sprite's flags
        Changed: renamed mapdraw() to map() for consistency
        Changed: default sleep time is 5ms (better cpu consumption for laptops)
        Fixed: memory limiter
        Fixed: wonky line and circle drawing
        Fixed: shift-click volume in sfx editor to set all
        Fixed: number formatting is now never in scientific notation
        Fixed: clipped error messages in console
        Fixed: text undo stores rollback points when chaning line number
        Fixed: print(str) carriage returns to previous x


v0.0.5

        Added: help()
        Added: Ctrl+F / Ctrl+G to search for text, repeat search
        Added: del key in code editor
        Added: Short-hand single-line IF statements
        Added: Unary operators += -= /= *= %=
        Added: srand(), time(), added rnd() to docs
        Added: Ctrl+D to duplicate line
        Added: interactive ls() for multi-page file listings
        Added: band() bor() bxor() bnot() shl() shr()
        Added: runtime error line number
        Added: dir() (aliased to ls())
        Changed: print() only autoscrolls when called with no parameters
        Changed: alt+up/down to skip between function definitions (was ctrl)
        Changed: sspr() dw, dh defaults to sw, sh
        Fixed: Load crashes on files that are not .p8 format or directories
        Fixed: Misc editor cursor position glitches
        Fixed: Crash when syntax error occurs before viewing code
        Fixed: Broken newlines after rebooting
        Fixed: mkdir() called with no parameters creating "(null)" directory
        Fixed: scrolling past top of code with scrollwheel
        Fixed: alt-f4 to fastquit


v0.0.4

        Added: Jelpi demo cart
        Added: Internal carts // use install_demos()
        Added: Joystick support
        Added: Undo/redo in code editor
        Added: Scroll wheel in code editor
        Added: LCTRL + UP/DOWN to navigate functions in code editor
        Added: LALT + LEFT/RIGHT to switch editing modes
        Added: btnp()
        Added: Release looping sample (a in editor , sfx(-2, channel) in code)
        Changed: Music stops when pausing program execution
        Changed: Allow 8 settable sprite flags
        Changed: Made noise instrument more bassy
        Fixed: Home, end keys
        Fixed: Sprite flags 4,5 not saved
        Fixed: mset() discarding 4 high bits
        Fixed: Crash when highlighting long strings


v0.0.3

        Added: Palette mapping type 1 (on display)
        Added: Collections can be initialized with c={1,2,..}
        Added: holdframe() // used automatically by _draw(), update()
        Added: Sprite selections and operations across selections
        Added: Map selection and stamp tool
        Added: Immediate mode screen buffer preserved while switching views
```

```
        Added: Channel mask for music playback
        Added: Memory mapping for live sound data
        Added: .png cart format
        Added: Sprite navigation by keyboard (-, +)
        Fixed: Strict 4-channel sound
        Fixed: Automatic sfx channel selection (channel index: -1)


v0.0.2

        Added: Command history
        Added: P2 keys
        Added: Boot sequence
        Added: Windows, 64-bit linux builds
        Added: CPU cost of internal api functions
        Added: Separate song channel index and mute status
        Added: Memory mapping
        Added: Search/replace colour in sprite editor
        Added: Copy/paste sprites and map regions
        Improved: Immediate mode command editing
        Improved: Editor cursor behaviour
        Fixed: Automatic audio channel selection


v0.0.1

        First Alpha
```