Zero Calculator

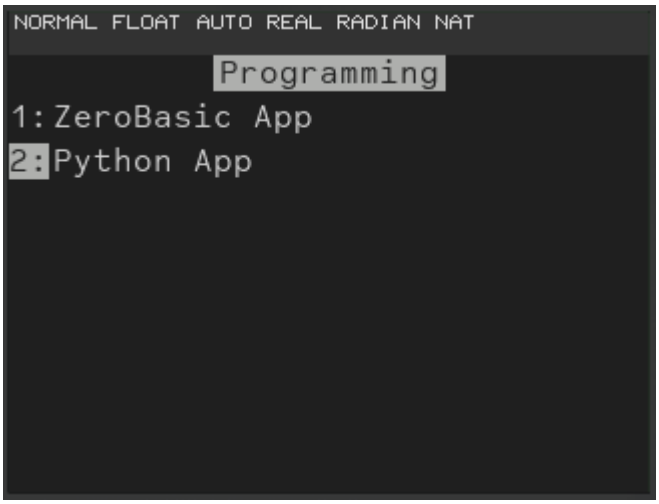
MicroPython User Manual (v2.27.1 dated 30.12.2025)

MicroPython is an implementation of the Python language written in C and designed to run on microcontrollers. More information about the MicroPython project can be found on the project website[1], in the project documentation,[2] and in the project repository[3].
Zero Calculator is able to interpret the MicroPython language. MicroPython is highly platform-dependent. This affects the modules and language features available for this platform. The language functions and features available to the user are described in the MicroPython Specification section.

I. Python app description

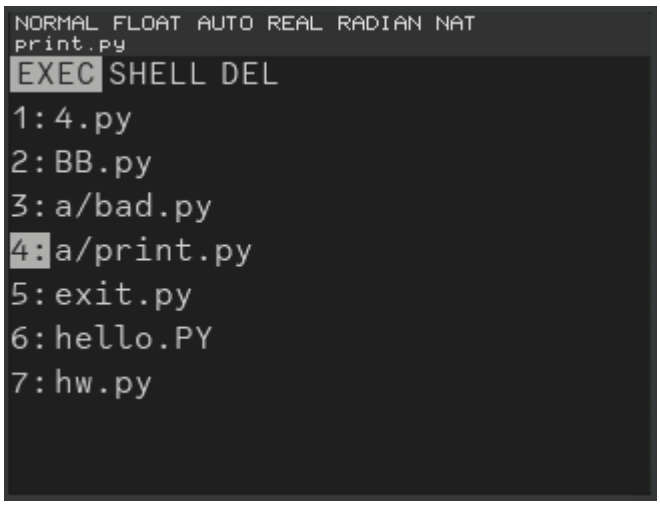
Python app is a separate Zero Calculator mode that contains its own screens and keyboard processing different from other modes.

To enter Python app, press the `prgm` key, then select Python App (`2`).

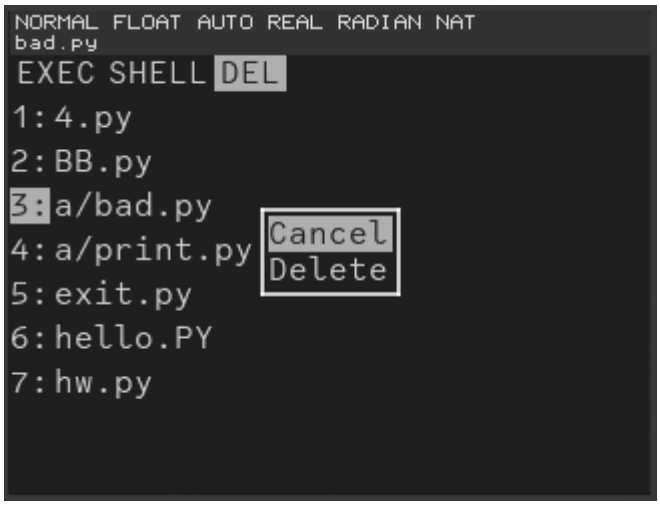


Python app contains three tabs (use `◄` and `►` to switch between tabs): **EXEC**, **SHELL**, **DEL**.

1. The **EXEC** tab contains a sorted list of `.py` files located in the `/exchange` folder. Files can also be located in subfolders. A file can be selected by using `▲` and `▼`. The name of the selected file is displayed in the tooltip in the header of the screen. Pressing `enter` or the button corresponding to the file activates the execution of the selected file in Python Shell window.



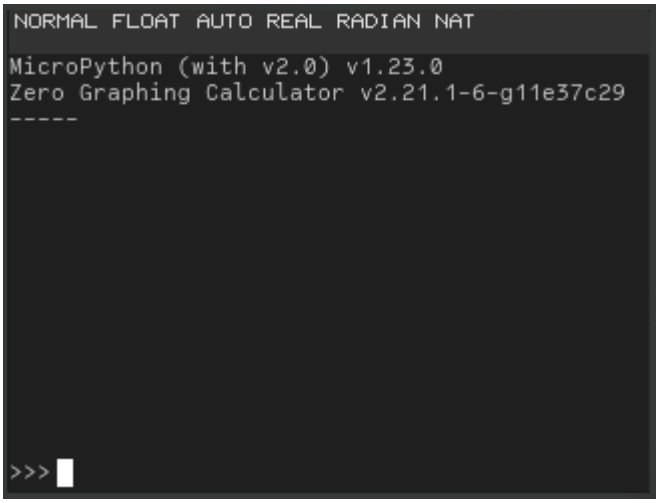
2. The **SHELL** tab. By pressing `enter` in this tab, the user can enter Python Shell window.
3. The **DEL** tab contains the same list of files as the **EXEC** tab. File activation in this tab opens the selection confirmation menu: **Cancel** (cancel deletion) and **Delete** (delete the selected file). Use `▲` and `▼` to select an action and `enter` to activate it. Closing the menu (by using `clear`, `quit` or other buttons) is equivalent to activating **Cancel**.



To exit from Python app, press `clear`, `quit`, or buttons that open other windows (not related to Python app).

II. Python Shell window

Python Shell window contains a history block and a command input block.



Initially, the history block contains a welcome message, which includes MicroPython header with its version and the platform header with its version. The text of the commands entered by the user along with the interpreter's interactive prompt and the text of the command execution result are saved to the history. The history can contain up to **2048** symbols. As the history is filled in, older content is truncated. The maximum count of visible lines in the history block is **15**. The maximum count of symbols in one line is **45**. If the string is longer than specified, then its content will be transferred to the following lines. The history block has vertical scrolling. Scrolling is performed by using `alpha` + `▲` and `alpha` + `▼`. The amount of scrolling is 3 lines. Also, adding information to the history block automatically scrolls it to the end.
The command input block is located at the bottom of the screen. It is used to enter commands for the Python interpreter. Command execution is started after pressing `enter`. After this, the control over the platform will

be passed to the Python interpreter. This means that the calculator screen will not be updated (except the `print()` command), the keyboard will not respond, and the internal processes of the calculator will stop until the command is interpreted. This implementation is needed to maximize the speed of command interpretation. Interrupting command interpretation can be done with on or physical platform reset (the reset button on the backside of the calculator). Command executionplaces its text to the command history (except for an empty command). The command history can contain up to **20** commands. Pressing ▲ extracts the previous command from the command history and places it to the command input block. If there is no previous command, the button is ignored. Pressing ▼ extracts the next command from the command history and places it to the command input block. If there is no next command, then an empty string is placed to the command input block. The command input block has horizontal scrolling. Scrolling is done automatically by trying to move the cursor beyond the input field by using ◄ and ►.

Commands entered by the user are executed in the same context of the Python interpreter. The context and the history of commands and results are reset after closing Python Shell window or shutting down the platform.

Keyboard processing in Python Shell window is different from keyboard processing in other modes of Zero Calculator. Below is the list of buttons, symbols, combinations, and the logic of processing thereof:

- 2nd, alpha work similarly to other Zero Calculator modes.
- on throws `KeyboardInterrupt` interruption at the moment of interpreting the command.
- ▲, ►, ▼, ◄ Their work is described above.
- 2nd + ◄, 2nd + ► moves the cursor to the beginning or end of the command input block.
- alpha + ▲, alpha + ▼ scrolls the history block.
- `quit` exits Python Shell window and switches to Python app.
- `off` switches off Zero Calculator.
- del removes previous symbol.
- clear clears command input block.
- ^, ,, (,), ÷, ×, -, +, ., 0 - 9, {, },], [, u, v, w, ", ?, : insert the corresponding character into the command input block.
- `A - Z` insert the corresponding case-sensitive letter into the command input block.
- sto→ inserts `=` into the command input block.
- (-) inserts `_` into the command input block.
- π inserts `pi` into the command input block.
- 2nd + ÷ inserts `\` into the command input block.
- 2nd + + inserts `'` into the command input block.
- 2nd + sto→ inserts `!` into the command input block.
- ␣ inserts space into the command input block.
- θ inserts `@` into the command input block.

Buttons, symbols, and combinations that are not included in this list are not processed in Python Shell window. The physical keyboard of the platform does not allow for entering certain characters (`~`, `#`, `$`, `%`, `&`, `<`, `>`, `|`), but the Python interpreter is able to process them. These characters can only be entered in the Zero Calculator emulator using a physical keyboard.

III. MicroPython Specification

Zero Calculator includes **MicroPython v1.23.0 dated 05.31.2024** which is built for the ARM Cortex-M4 architecture (without emulators). This build implements Python 3.4 and some featured functions of Python 3.5 and above (comparison[4]). This MicroPython build has the following building options (the options order is set by the configuration file mpconfig.h[5]):

- [Enabled] In-progress/breaking changes slated for the MicroPython 2.x release (`MICROPY_PREVIEW_VERSION_2`).
- Used minimal port with disabling all optional features (`MICROPY_CONFIG_ROM_LEVEL` = `MICROPY_CONFIG_ROM_LEVEL_MINIMUM`).
- Default allocation memory settings (`MICROPY_ALLOC_...` , `MICROPY_GC_...` , `MICROPY_OBJ_...`).
- [Enabled] Passing allocated memory region size to realloc/free functions (`MICROPY_MALLOC_USES_ALLOCATED_SIZE`).
- 16 bytes allocate initially when creating new chunks to store parse nodes (`MICROPY_ALLOC_PARSE_CHUNK_INIT`).
- Maximum length of a path in the filesystem is 256 symbols (`MICROPY_ALLOC_PATH_MAX`).
- [Disabled] Supporting loading of persistent code (mpy) (default) (`MICROPY_PERSISTENT_CODE_LOAD`).
- [Enabled] Ability to interpret Python commands (`MICROPY_ENABLE_COMPILER`).
- [Disabled] Optimizations and calculations during compilation (minimal port) (`MICROPY_COMP_CONST_FOLDING` , `MICROPY_COMP_CONST_TUPLE` , `MICROPY_COMP_CONST_LITERAL` , `MICROPY_COMP_MODULE_CONST` , `MICROPY_COMP_CONST` , `MICROPY_COMP_DOUBLE_TUPLE_ASSIGN` , `MICROPY_COMP_TRIPLE_TUPLE_ASSIGN` , `MICROPY_COMP_RETURN_IF_EXPR`).
- [Enabled] Collect memory allocation stats (functions `micropython.mem_total` , `micropython.mem_current` , `micropython.mem_peak`)(`MICROPY_MEM_STATS`).
- [Disabled] Debug settings (default) (`MICROPY_DEBUG_PRINTERS` , `MICROPY_DEBUG_VERBOSE` , `MICROPY_DEBUG_MP_OBJ_SENTINELS` , `MICROPY_DEBUG_PARSE_RULE_NAME` , `MICROPY_DEBUG_VM_STACK_OVERFLOW` , `MICROPY_DEBUG_VALGRIND`).
- [Disabled] Optimizations (default, minimal port) (`MICROPY_OPT_COMPUTED_GOTO` , `MICROPY_OPT_LOAD_ATTR_FAST_PATH` , `MICROPY_OPT_MAP_LOOKUP_CACHE` , `MICROPY_OPT_MAP_LOOKUP_CACHE_SIZE` , `MICROPY_OPT_MPZ_BITWISE` , `MICROPY_OPT_MATH_FACTORIAL`).
- [Enabled] Import of external modules (files) (`MICROPY_ENABLE_EXTERNAL_IMPORT`).
- [Enabled] File reader for importing files (`MICROPY_READER_VFS`).
- [Enabled] Garbage collector with default settings (`MICROPY_ENABLE_GC`).
- [Enabled] Calling finalisers in the garbage collector (`__del__`) (`MICROPY_ENABLE_FINALISER`).
- [Disabled] Separate allocator for the Python stack (default) (`MICROPY_ENABLE_PYSTACK`).
- [Disabled] Emergency exception buffer (default) (`MICROPY_ENABLE_EMERGENCY_EXCEPTION_BUF`).
- [Enabled] Keyboard interrupt (`MICROPY_KBD_EXCEPTION`).

- [Enabled] Input helper functions (`MICROPY_HELPER_REPL`).
- Allow enabling debug prints after each entered command (`MICROPY_REPL_INFO`).
- [Disabled] Auto indent (`MICROPY_REPL_AUTO_INDENT`).
- [Enabled] Event-driven input functions (`MICROPY_REPL_EVENT_DRIVEN`).
- Count of commands keeped in Python history: 0 (`MICROPY_READLINE_HISTORY_SIZE`).
- [Disabled] Terminal VT100 (`MICROPY_HAL_HAS_VT100`).
- `long int` is an arbitrary precision integer type (`MICROPY_LONGINT_IMPL = MICROPY_LONGINT_IMPL_MPZ`).
- [Disabled] Processing line numbers of the source (minimal port) (`MICROPY_ENABLE_SOURCE_LINE`).
- [Disabled] Doc strings (`__doc__`) (default), (`MICROPY_ENABLE_DOC_STRING`).
- [Enabled] Print basic error and exception details (`MICROPY_ERROR_REPORTING = MICROPY_ERROR_REPORTING_NORMAL`).
- [Disabled] Warnings (default) (`MICROPY_WARNINGS` , `MICROPY_WARNINGS_CATEGORY`).
- `float` is `double` (`MICROPY_FLOAT_IMPL = MICROPY_FLOAT_IMPL_DOUBLE`).
- [Enabled] Complex numbers (`MICROPY_PY_BUILTINS_COMPLEX`).
- [Disabled] High-quality hash for float and complex numbers (minimal port) (`MICROPY_FLOAT_HIGH_QUALITY_HASH`).
- [Disabled] Features which improve CPython compatibility (minimal port) (`MICROPY_CPYTHON_COMPAT`).
- [Disabled] Full checks as done by CPython (minimal port) (`MICROPY_FULL_CHECKS`).
- [Disabled] POSIX-semantics non-blocking streams (default, minimal port) (`MICROPY_STREAMS_NON_BLOCK` , `MICROPY_STREAMS_POSIX_API`).
- [Disabled] Calling `__init__` when importing builtin modules for the first time (minimal port) (`MICROPY_MODULE_BUILTIN_INIT`).
- [Disabled] Built-in modules having sub-packages (minimal port) (`MICROPY_MODULE_BUILTIN_SUBPACKAGES`).
- [Disabled] Module-level `__getattr__` (minimal port) (`MICROPY_MODULE_GETATTR`).
- [Disabled] Setting `__name__` to `'__main__'` when importing file (default) (`MICROPY_MODULE_OVERRIDE_MAIN_IMPORT`).
- [Disabled] Frozen modules (default) (`MICROPY_MODULE_FROZEN`).
- [Disabled] Overriding builtins in the builtins module (minimal port) (`MICROPY_CAN_OVERRIDE_BUILTINS`).
- [Enabled] Checking that the `self` argument of a builtin method has the correct type (`MICROPY_BUILTIN_METHOD_CHECK_SELF_ARG`).
- [Enabled] Using internally defined errno's (`MICROPY_USE_INTERNAL_ERRNO`).
- [Disabled] Using internally defined `*printf()` functions (`MICROPY_USE_INTERNAL_PRINTF`).
- [Disabled] Asynchronously aborting to the top level (default) (`MICROPY_ENABLE_VM_ABORT`).
- [Disabled] Internal scheduler (minimal port) (`MICROPY_ENABLE_SCHEDULER` , `MICROPY_SCHEDULER_STATIC_NODES`).
- [Enabled] Virtual file system (more in Platform features and abilities) (`MICROPY_VFS`).

- [Disabled] Multiple inheritance of Python classes (minimal port) (`MICROPY_MULTIPLE_INHERITANCE`).
- [Enabled] Implementing attributes on functions (`MICROPY_PY_FUNCTION_ATTRS`).
- [Disabled] Descriptors `__get__` , `__set__` , `__delete__` (minimal port) (`MICROPY_PY_DESCRIPTORS`).
- [Disabled] Class `__delattr__` , `__setattr__` methods (minimal port) (`MICROPY_PY_DELATTR_SETATTR`).
- [Disabled] Async/await (minimal port) (`MICROPY_PY_ASYNC_AWAIT`).
- [Enabled] F-strings (`MICROPY_PY_FSTRINGS`).
- [Disabled] Assignment expressions with `:=` (minimal port) (`MICROPY_PY_ASSIGN_EXPR`).
- [Disabled] Non-standard `.pend_throw()` method for generators (minimal port) (`MICROPY_PY_GENERATOR_PEND_THROW`).
- [Disabled] Warning when comparing str and bytes objects (default) (`MICROPY_PY_STR_BYTES_CMP_WARN`).
- [Disabled] Methods `bytes.hex` and `bytes.fromhex` (minimal port) (`MICROPY_PY_BUILTINS_BYTES_HEX`).
- [Disabled] Unicode strings (minimal port) (`MICROPY_PY_BUILTINS_STR_UNICODE`).
- [Disabled] Check for valid UTF-8 when converting bytes to str (minimal port) (`MICROPY_PY_BUILTINS_STR_UNICODE_CHECK`).
- [Enabled] Method `str.count()` (`MICROPY_PY_BUILTINS_STR_COUNT`).
- [Enabled] Operator `str % (...)` (`MICROPY_PY_BUILTINS_STR_OP_MODULO`).
- [Disabled] Methods `str.center()` , `str.partition()` , `str.rpartition()` , `str.splitlines()` (minimal port) (`MICROPY_PY_BUILTINS_STR_CENTER` , `MICROPY_PY_BUILTINS_STR_PARTITION` , `MICROPY_PY_BUILTINS_STR_SPLITLINES`).
- [Enabled] Type `bytearray` (`MICROPY_PY_BUILTINS_BYTEARRAY`).
- [Disabled] Methods `dict.fromkeys()` (minimal port) (`MICROPY_PY_BUILTINS_DICT_FROMKEYS`).
- [Enabled] Type `memoryview` (`MICROPY_PY_BUILTINS_MEMORYVIEW`).
- [Disabled] Property `memoryview.itemsize` (minimal port) (`MICROPY_PY_BUILTINS_MEMORYVIEW_ITEMSIZE`).
- [Enabled] Type `set` (`MICROPY_PY_BUILTINS_SET`).
- [Enabled] `slice` (`[a:b]`) (`MICROPY_PY_BUILTINS_SLICE`).
- [Disabled] Properties `slice.start` , `slice.stop` , `slice.step` (minimal port) (`MICROPY_PY_BUILTINS_SLICE_ATTRS`).
- [Disabled] Methods `slice.indices(len)` (minimal port) (`MICROPY_PY_BUILTINS_SLICE_INDICES`).
- [Disabled] Type `frozenset` (minimal port) (`MICROPY_PY_BUILTINS_FROZENSET`).
- [Disabled] Type `property` (minimal port) (`MICROPY_PY_BUILTINS_PROPERTY`).
- [Disabled] Properties `range.start` , `range.stop` , `range.step` (minimal port) (`MICROPY_PY_BUILTINS_RANGE_ATTRS`).
- [Disabled] Comparing (inequality/equality) `range` objects (minimal port) (`MICROPY_PY_BUILTINS_RANGE_BINOP`).
- [Disabled] Calling `next()` with second argument (minimal port) (`MICROPY_PY_BUILTINS_NEXT2`).

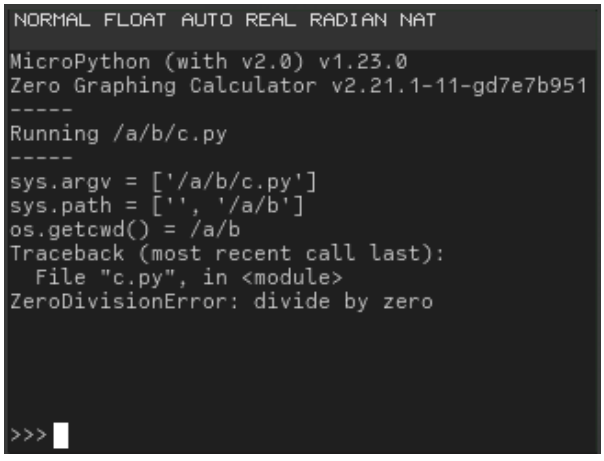
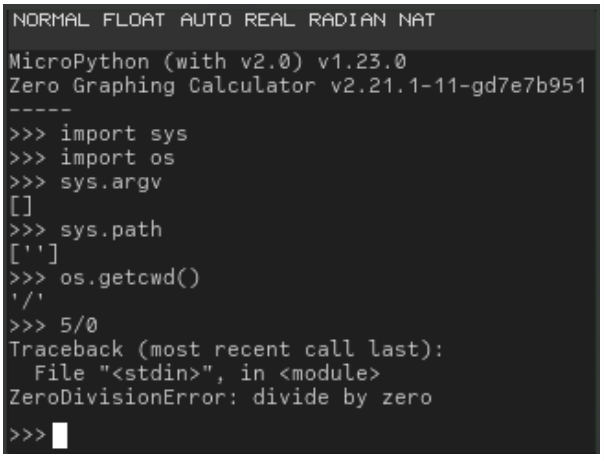
- [Enabled] Function `round(int, int)` (`MICROPY_PY_BUILTINS_ROUND_INT`).
- [Disabled] Supporting complete set of special methods for user classes (minimal port) (`MICROPY_PY_ALL_SPECIAL_METHODS`).
- [Disabled] Supporting all inplace arithmetic operation methods for user classes (`__imul__` and another) (minimal port) (`MICROPY_PY_ALL_INPLACE_SPECIAL_METHODS`).
- [Disabled] Supportingreverse arithmetic operation methods for user classes (`__radd__` and another) (minimal port) (`MICROPY_PY_REVERSE_SPECIAL_METHODS`).
- [Disabled] Function `compile()` (minimal port) (`MICROPY_PY_BUILTINS_COMPILE`).
- [Enabled] Function and type `enumerate()` (`MICROPY_PY_BUILTINS_ENUMERATE`).
- [Enabled] Functions `eval` , `exec` (`MICROPY_PY_BUILTINS_EVAL_EXEC`).
- [Disabled] Python 2 function `execfile()` (minimal port) (`MICROPY_PY_BUILTINS_EXECFILE`).
- [Disabled] Function `filter()` (minimal port) (`MICROPY_PY_BUILTINS_FILTER`).
- [Enabled] Function `reversed()` (`MICROPY_PY_BUILTINS_REVERSED`).
- [Disabled] Constant `"NotImplemented"` (minimal port) (`MICROPY_PY_BUILTINS_NOTIMPLEMENTED`).
- [Disabled] Function `input()` (`MICROPY_PY_BUILTINS_INPUT`).
- [Enabled] Functions `min` , `max` (`MICROPY_PY_BUILTINS_MIN_MAX`).
- [Disabled] Function `pow(int, int, int)` (minimal port) (`MICROPY_PY_BUILTINS_POW3`).
- [Enabled] Function `help()` (`MICROPY_PY_BUILTINS_HELP` , `MICROPY_PY_BUILTINS_HELP_TEXT` , `MICROPY_PY_BUILTINS_HELP_MODULES`).
- [Disabled] Variable `__FILE__` (minimal port) (`MICROPY_PY___FILE__`).
- [Enabled] Functions `micropython.mem_total` , `micropython.mem_current` , `micropython.mem_peak` , `micropython.stack_use` (`MICROPY_PY_MICROPYTHON_MEM_INFO` , `MICROPY_PY_MICROPYTHON_STACK_USE`).
- [Disabled] Function `micropython.heap_locked` (minimal port) (`MICROPY_PY_MICROPYTHON_HEAP_LOCKED`).
- [Enabled] Module `array` (`MICROPY_PY_ARRAY`).
- [Disabled] Slice assignments for array (`a[0:2] = b`) (minimal port) (`MICROPY_PY_ARRAY_SLICE_ASSIGN`).
- [Enabled] Type `attrtuple` (space-efficient `namedtuple`) (`MICROPY_PY_ATTRTUPLE`).
- [Enabled] Module `collections` (`MICROPY_PY_COLLECTIONS`).
- [Disabled] Types `collections.deque` , `collections.OrderedDict` (minimal port) (`MICROPY_PY_COLLECTIONS_DEQUE` , `MICROPY_PY_COLLECTIONS_ORDEREDDICT`).
- [Disabled] Methods `namedtuple._asdict()` (minimal port) (`MICROPY_PY_COLLECTIONS_NAMEDTUPLE__ASDICT`).
- [Enabled] Module `math` (`MICROPY_PY_MATH`).
- [Disabled] Addition math module constants without `math.pi` , `math.e` (minimal port) (`MICROPY_PY_MATH_CONSTANTS`).
- [Disabled] Addition math module functions `math.erf` , `math.erfc` , `math.gamma` , `math.lgamma` , `math.factorial` , `math.isclose` (minimal port) (`MICROPY_PY_MATH_SPECIAL_FUNCTIONS` , `MICROPY_PY_MATH_SPECIAL_FUNCTIONS` , `MICROPY_PY_MATH_FACTORIAL` , `MICROPY_PY_MATH_ISCLOSE`).

- [Disabled] Fixes for math module functions `math.atan2`, `math.fmod`, `math.modf`, `math.pow` (default) (`MICROPY_PY_MATH_ATAN2_FIX_INFNAN`, `MICROPY_PY_MATH_FMOD_FIX_INFNAN`, `MICROPY_PY_MATH_MODF_FIX_NEGZERO`, `MICROPY_PY_MATH_POW_FIX_NAN`).
- [Enabled] Module `cmath` (`MICROPY_PY_CMATH`).
- [Enabled] Module `micropython` (`MICROPY_PY_MICROPYTHON`).
- [Enabled] Module `gc` (`MICROPY_PY_GC`).
- [Enabled] Module `io` (`MICROPY_PY_IO`).
- [Disabled] Classes `io.IOBase`, `io.BufferedWriter` (minimal port) (`MICROPY_PY_IO_IOBASE`, `MICROPY_PY_IO_BUFFEREDWRITER`).
- [Enabled] Module `struct` (`MICROPY_PY_STRUCT`).
- [Enabled] Module `sys` (`MICROPY_PY_SYS`).
- [Disabled] Constants, properties and functions `sys.maxsize`, `sys.exc_info`, `sys.executable`, `sys.intern`, `sys.atexit`, `sys.ps1`, `sys.ps2`, `sys.settrace`, `sys.getsizeof`, `sys.stdin`, `sys.stdout`, `sys.stderr`, `sys.tracebacklimit` (default, minimal port) (`MICROPY_PY_SYS_MAXSIZE`, `MICROPY_PY_SYS_EXC_INFO`, `MICROPY_PY_SYS_EXECUTABLE`, `MICROPY_PY_SYS_INTERN`, `MICROPY_PY_SYS_ATEXIT`, `MICROPY_PY_SYS_PS1_PS2`, `MICROPY_PY_SYS_SETTRACE`, `MICROPY_PY_SYS_GETSIZEOF`, `MICROPY_PY_SYS_STDFILES`, `MICROPY_PY_SYS_STDIO_BUFFER`, `MICROPY_PY_SYS_TRACEBACKLIMIT`).
- [Enabled] Dictionary `sys.modules` (default) (`MICROPY_PY_SYS_MODULES`).
- [Enabled] Function `sys.exit` (default) (`MICROPY_PY_SYS_EXIT`).
- [Enabled] Lists `sys.path`, `sys.argv` (`MICROPY_PY_SYS_PATH`, `MICROPY_PY_SYS_ARGV`).
- [Enabled] Module `errno` (`MICROPY_PY_ERRNO`).
- [Enabled] Dictionary `errno.errorcode` (default) (`MICROPY_PY_ERRNO_ERRORCODE`).
- [Disabled] Module `select` (minimal port) (`MICROPY_PY_SELECT`).
- [Disabled] Module `time` (minimal port) (`MICROPY_PY_TIME`).
- [Disabled] Module `_thread` (`MICROPY_PY_THREAD`).
- [Disabled] Module `asyncio` (minimal port) (`MICROPY_PY_ASYNCIO`).
- [Disabled] Module `uctypes` (minimal port) (`MICROPY_PY_UCTYPES`).
- [Disabled] Module `deflate` (minimal port) (`MICROPY_PY_DEFLATE`).
- [Disabled] Module `json` (minimal port) (`MICROPY_PY_JSON`).
- [Enabled] Module `os` (`MICROPY_PY_OS`).
- [Disabled] Function `os.statvfs` (`MICROPY_PY_OS_STATVFS`).
- [Enabled] Function `os.uname` (`MICROPY_PY_OS_UNAME`).
- [Enabled] Constant `os.sep` (`MICROPY_PY_OS_SEP`).
- [Disabled] Module `re` (minimal port) (`MICROPY_PY_RE`).
- [Disabled] Module `heapq` (minimal port) (`MICROPY_PY_HEAPQ`).
- [Disabled] Module `hashlib` (minimal port) (`MICROPY_PY_HASHLIB`).
- [Disabled] Module `cryptolib` (minimal port) (`MICROPY_PY_CRYPTOLIB`).
- [Disabled] Module `binascii` (minimal port) (`MICROPY_PY_BINASCII`).
- [Disabled] Module `random` (minimal port) (`MICROPY_PY_RANDOM`).
- [Disabled] Module `machine` (`MICROPY_PY_MACHINE`).

- [Disabled] Module `lwip` (`MICROPY_PY_LWIP`).
- [Disabled] Module `ssl` (default) (`MICROPY_PY_SSL`).
- [Disabled] Module `vfs` (`MICROPY_PY_VFS`).
- [Disabled] Module `websocket` (default) (`MICROPY_PY_WEBSOCKET`).
- [Disabled] Module `framebuf` (minimal port) (`MICROPY_PY_FRAMEBUF`).
- [Disabled] Module `btree` (default) (`MICROPY_PY_BTREE`).
- [Disabled] Module `_onewire` (default) (`MICROPY_PY_ONEWIRE`).
- [Disabled] Module `platform` (minimal port) (`MICROPY_PY_PLATFORM`).

3.1. Platform features and abilities

Regardless of the running of the script file or Python Shell, a heap area is allocated for MicroPython interpreter. The heap area size is equal to half the volume of all free RAM, but no more than **4MB** (4194304 bytes). MicroPython interpreter can manage a virtual file system. The virtual filesystem root folder (`/`) is at `/exchange` . A recursive script search in **EXEC** tab of Python app is also performed through this path. The `io` and `os` modules can interact with the file system and files. Running a script file or Python Shell has some differences:

Running `/a/b/c.py`	**Running Python Shell**
Welcome message: `Running /a/b/c.py`	
`sys.argv = ['/a/b/c.py']`	`sys.argv = []`
`sys.path = ['','/a/b']`	`sys.path = ['']`
`os.getcwd() = '/a/b'`	`os.getcwd() = '/'`
First traceback level: `c.py`	First traceback level: `<stdin>`
	

The implementation of the following modules depends on the current platform:

- `sys` (more[6]), implemented:
  - `argv` <list>
  - `path` <list>
  - `version` <string>
  - `version_info` <tuple>

    - `implementation` <namedtuple>
    - `platform` <string>
    - `byteorder` <string>
    - `exit` <function> (Calling this function will exit Python Shell window and return to the main screen of Zero Calculator. The argument passed to the function will be converted into a string and inserted into the editor.)
    - `modules` <dict>
    - `print_exception` <function>
- `os` (more[7]), implemented:
    - `sep` <string>
    - `uname` <function>
    - `chdir` <function>
    - `getcwd` <function> (current directory may not exist if you delete it after switching to it)
    - `listdir` <function> (if argument is not passed, outputs the contents of the current catalog `.`)
    - `mkdir` <function> (also creates child directories if they don't exist)
    - `remove` <function> (deletes the file if it exists)
    - `rename` <function> (movement is performed only to existing paths)
    - `rmdir` <function> (deletes an empty directory if it exists)
    - `stat` <function> (returns <tuple>, 0th item (`st_mode`): 0x8000 - file, 0x4000 - directory, 6th item (`st_size`): file size in bytes (directory size is 0), 8th item (`st_mtime`): file change timestamp (Zero Calculator uses FAT file system), other items are 0).
    - `statvfs` <function> (throws an exception when used)
    - `unlink` <function> (deletes the file if it exists)
    - `ilistdir` <function>
- `errno` (more[8]), implemented:
    - `errorcode` <dict>
    - `EPERM` <int> (operation not permitted)
    - `ENOENT` <int> (no such file or directory)
    - `EIO` <int> (i/o error)
    - `EBADF` <int> (bad file number)
    - `EAGAIN` <int> (try again)
    - `ENOMEM` <int> (out of memory)
    - `EACCES` <int> (permission denied)
    - `EEXIST` <int> (file exists)
    - `ENODEV` <int> (no such device)
    - `ENOTDIR` <int> (not a directory)
    - `EISDIR` <int> (is a directory)
    - `EINVAL` <int> (invalid argument)
    - `EROFS` <int> (read-only file system)
    - `EOPNOTSUPP` <int> (operation not supported on transport endpoint)
    - `EADDRINUSE` <int> (address already in use)
    - `ECONNABORTED` <int> (software caused connection abort)
    - `ECONNRESET` <int> (connection reset by peer)
    - `ENOBUFS` <int> (no buffer space available)

    - `ENOTCONN` <int> (transport endpoint is not connected)
    - `ETIMEDOUT` <int> (connection timed out)
    - `ECONNREFUSED` <int> (connection refused)
    - `EHOSTUNREACH` <int> (no route to host)
    - `EALREADY` <int> (operation already in progress)
    - `EINPROGRESS` <int> (operation now in progress)
    - `EFAILED` <int> (Zero Calculator internal error, it is mainly used if the file system cannot process the request)
- `micropython` (more[9]) , implemented:
    - `const` <function>
    - `opt_level` <function>
    - `mem_total` <function>
    - `mem_current` <function>
    - `mem_peak` <function>
    - `mem_info` <function>
    - `qstr_info` <function>
    - `stack_use` <function>
    - `heap_lock` <function>
    - `heap_unlock` <function>
    - `kbd_intr` <function> (is not processed because Zero Calculator uses internal processing of keyboard interruption)

To get more detailed information about the available data types, functions, methods, and modules, use the function `help()`. Also refer to the online documentation of Python[10] and MicroPython[2] (modules description[11], differences from Python[4]).

IV. LVGL Library

Micropython support using the LVGL (LittlevGV) library version v6.1.2.

LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

To get more information about the LVGL library refer to the online documentation of LVLG[12]

This documentation provides both basic library information and detailed descriptions with examples for each type of graphical object.

4.1. Zero Micropython + LVGL implementation

To work with LVGL library you need to import it by calling the command `import lvgl`. The base (parent) object for drawing is the screen. Access to this object can be obtained by calling the `scr_act` function. This object occupies the entire display excluding the header. It is forbidden to delete this object, change its position or size. However, you can use your own design functions with this object (an example will be given below). This screen is hidden by default and it is possible to use `show` and `hide` functions to show it. When the Python script execution is finished, but the draw screen is still open, it is possible to close it using the **`quit`** key. Interrupting script execution is also available by pressing the on key. The rest of the information on LVGL can be found at the provided link to the LVGL documentation and the examples below. The Python examples in the official documentation have a slightly different API, so it is recommended to use the examples in this document. You can also use the `help` function to get information on LVGL.

4.1.1. Keyboard support from LVGL

To support keyboard input, register an LVGL `indev` device with type `KEYPAD` and read callback `lvgl.keypad_read`. Graphical objects that are to be controlled from the keyboard must be added to a `group`, and the `group` is bound to the `indev`. Below is a table of how the physical calculator buttons are bound to the LVGL input device:

Phisical key	LVGL key	Action
▲	`LV_KEY_UP`	Increase value or move upwards
▼	`LV_KEY_DOWN`	Decrease value or move downwards
▶	`LV_KEY_RIGHT`	Increase value or move the the right
◀	`LV_KEY_LEFT`	Decrease value or move the the left
del	`LV_KEY_BACKSPACE`	Delete a character on the left
clear	`LV_KEY_ESC`	Close or exit
enter	`LV_KEY_ENTER`	Triggers LV_EVENT_PRESSED/CLICKED events
alpha + ▲	`LV_KEY_HOME`	Go to the beginning/top (E.g. in a Text area)
alpha + ▼	`LV_KEY_END`	Go to the end (E.g. in a Text area)

Phisical key	LVGL key	Action
2nd + ◄	`LV_KEY_PREV`	Focus on the previous object
2nd + ►	`LV_KEY_NEXT`	Focus on the next object

Read more information about input devices and LVGL groups in the LVGL documentation[13]

The creation and use of input devices is shown in some examples below.

4.2. Zero Micropython module objects overview

- `keypad_read` -- <mp_keypad_read>
- `show` -- <function>
- `hide` -- <function>
- `get_header_height` -- <function>
- `__name__` -- lvgl
- `obj` -- <class 'obj'>
- `cont` -- <class 'cont'>
- `btn` -- <class 'btn'>
- `imgbtn` -- <class 'imgbtn'>
- `label` -- <class 'label'>
- `img` -- <class 'img'>
- `line` -- <class 'line'>
- `page` -- <class 'page'>
- `list` -- <class 'list'>
- `chart` -- <class 'chart'>
- `table` -- <class 'table'>
- `cb` -- <class 'cb'>
- `cpicker` -- <class 'cpicker'>
- `bar` -- <class 'bar'>
- `slider` -- <class 'slider'>
- `led` -- <class 'led'>
- `btnm` -- <class 'btnm'>
- `ddlist` -- <class 'ddlist'>
- `roller` -- <class 'roller'>
- `ta` -- <class 'ta'>
- `canvas` -- <class 'canvas'>
- `win` -- <class 'win'>
- `tabview` -- <class 'tabview'>
- `tileview` -- <class 'tileview'>
- `mbox` -- <class 'mbox'>
- `lmeter` -- <class 'lmeter'>
- `gauge` -- <class 'gauge'>
- `sw` -- <class 'sw'>
- `arc` -- <class 'arc'>

- `preload` -- <class 'preload'>
- `spinbox` -- <class 'spinbox'>
- `color_make` -- <function>
- `color_hex` -- <function>
- `color_hex3` -- <function>
- `style_anim_set_time` -- <function>
- `style_anim_set_ready_cb` -- <function>
- `style_anim_set_playback` -- <function>
- `style_anim_clear_playback` -- <function>
- `style_anim_set_repeat` -- <function>
- `style_anim_clear_repeat` -- <function>
- `style_anim_create` -- <function>
- `scr_act` -- <function>
- `mem_init` -- <function>
- `mem_deinit` -- <function>
- `mem_alloc` -- <function>
- `mem_free` -- <function>
- `mem_realloc` -- <function>
- `mem_defrag` -- <function>
- `mem_get_size` -- <function>
- `task_core_init` -- <function>
- `task_handler` -- <function>
- `task_create_basic` -- <function>
- `task_create` -- <function>
- `task_enable` -- <function>
- `task_get_idle` -- <function>
- `trigo_sin` -- <function>
- `bezier3` -- <function>
- `atan2` -- <function>
- `sqrt` -- <function>
- `async_call` -- <function>
- `color_hsv_to_rgb` -- <function>
- `color_rgb_to_hsv` -- <function>
- `tick_get` -- <function>
- `tick_elaps` -- <function>
- `anim_core_init` -- <function>
- `anim_del` -- <function>
- `anim_count_running` -- <function>
- `anim_speed_to_time` -- <function>
- `style_init` -- <function>
- `style_anim_init` -- <function>
- `style_anim_set_styles` -- <function>
- `init` -- <function>
- `event_send` -- <function>
- `event_send_func` -- <function>

- `event_get_data` -- <function>
- `signal_send` -- <function>
- `group_init` -- <function>
- `group_create` -- <function>
- `group_remove_obj` -- <function>
- `group_focus_obj` -- <function>
- `indev_init` -- <function>
- `indev_read_task` -- <function>
- `indev_get_act` -- <function>
- `indev_get_obj_act` -- <function>
- `debug_check_null` -- <function>
- `debug_check_obj_type` -- <function>
- `debug_check_obj_valid` -- <function>
- `debug_check_style` -- <function>
- `debug_check_str` -- <function>
- `debug_log_error` -- <function>
- `theme_get_current` -- <function>
- `txt_get_size` -- <function>
- `txt_get_next_line` -- <function>
- `txt_get_width` -- <function>
- `txt_is_cmd` -- <function>
- `txt_ins` -- <function>
- `txt_cut` -- <function>
- `draw_get_buf` -- <function>
- `draw_free_buf` -- <function>
- `draw_aa_get_opa` -- <function>
- `draw_aa_ver_seg` -- <function>
- `draw_aa_hor_seg` -- <function>
- `draw_px` -- <function>
- `draw_fill` -- <function>
- `draw_letter` -- <function>
- `draw_map` -- <function>
- `draw_rect` -- <function>
- `draw_label` -- <function>
- `draw_line` -- <function>
- `draw_triangle` -- <function>
- `draw_polygon` -- <function>
- `draw_arc` -- <function>
- `draw_img` -- <function>
- `RES` -- <class 'LV_RES'>
- `TASK_PRIO` -- <class 'LV_TASK_PRIO'>
- `OPA` -- <class 'LV_OPA'>
- `INDEV_TYPE` -- <class 'LV_INDEV_TYPE'>
- `INDEV_STATE` -- <class 'LV_INDEV_STATE'>
- `FONT_SUBPX` -- <class 'LV_FONT_SUBPX'>

- `ANIM` -- <class 'LV_ANIM'>
- `BORDER` -- <class 'LV_BORDER'>
- `SHADOW` -- <class 'LV_SHADOW'>
- `BIDI_DIR` -- <class 'LV_BIDI_DIR'>
- `DESIGN` -- <class 'LV_DESIGN'>
- `EVENT` -- <class 'LV_EVENT'>
- `SIGNAL` -- <class 'LV_SIGNAL'>
- `ALIGN` -- <class 'LV_ALIGN'>
- `DRAG_DIR` -- <class 'LV_DRAG_DIR'>
- `PROTECT` -- <class 'LV_PROTECT'>
- `KEY` -- <class 'LV_KEY'>
- `GROUP_REFOCUS_POLICY` -- <class 'LV_GROUP_REFOCUS_POLICY'>
- `FONT_FMT_TXT_CMAP` -- <class 'LV_FONT_FMT_TXT_CMAP'>
- `LAYOUT` -- <class 'LV_LAYOUT'>
- `FIT` -- <class 'LV_FIT'>
- `TXT_FLAG` -- <class 'LV_TXT_FLAG'>
- `TXT_CMD_STATE` -- <class 'LV_TXT_CMD_STATE'>
- `SB_MODE` -- <class 'LV_SB_MODE'>
- `CURSOR` -- <class 'LV_CURSOR'>
- `FONT_FMT_TXT` -- <class 'LV_FONT_FMT_TXT'>
- `SYMBOL` -- <class 'LV_SYMBOL'>
- `C_Pointer` -- <class 'C_Pointer'>
- `area_t` -- <class 'lv_area_t'>
- `style_t` -- <class 'lv_style_t'>
- `style_body_t` -- <class 'lv_style_body_t'>
- `color16_t` -- <class 'lv_color16_t'>
- `color16_ch_t` -- <class 'lv_color16_ch_t'>
- `style_body_border_t` -- <class 'lv_style_body_border_t'>
- `style_body_shadow_t` -- <class 'lv_style_body_shadow_t'>
- `style_body_padding_t` -- <class 'lv_style_body_padding_t'>
- `style_text_t` -- <class 'lv_style_text_t'>
- `font_t` -- <class 'lv_font_t'>
- `font_glyph_dsc_t` -- <class 'lv_font_glyph_dsc_t'>
- `style_image_t` -- <class 'lv_style_image_t'>
- `style_line_t` -- <class 'lv_style_line_t'>
- `task_t` -- <class 'lv_task_t'>
- `ll_t` -- <class 'lv_ll_t'>
- `obj_type_t` -- <class 'lv_obj_type_t'>
- `point_t` -- <class 'lv_point_t'>
- `img_header_t` -- <class 'lv_img_header_t'>
- `img_decoder_dsc_t` -- <class 'lv_img_decoder_dsc_t'>
- `img_decoder_t` -- <class 'lv_img_decoder_t'>
- `img_dsc_t` -- <class 'lv_img_dsc_t'>
- `img_cache_entry_t` -- <class 'lv_img_cache_entry_t'>
- `chart_series_t` -- <class 'lv_chart_series_t'>

- color_hsv_t -- <class 'lv_color_hsv_t'>
- _lv_mp_int_wrapper -- <class '_lv_mp_int_wrapper'>
- anim_t -- <class 'lv_anim_t'>
- mem_monitor_t -- <class 'lv_mem_monitor_t'>
- indev_drv_t -- <class 'lv_indev_drv_t'>
- indev_data_t -- <class 'lv_indev_data_t'>
- indev_t -- <class 'lv_indev_t'>
- indev_proc_t -- <class 'lv_indev_proc_t'>
- indev_proc_types_t -- <class 'lv_indev_proc_types_t'>
- indev_proc_types_pointer_t -- <class 'lv_indev_proc_types_pointer_t'>
- indev_proc_types_keypad_t -- <class 'lv_indev_proc_types_keypad_t'>
- group_t -- <class 'lv_group_t'>
- theme_t -- <class 'lv_theme_t'>
- theme_style_t -- <class 'lv_theme_style_t'>
- theme_style_btn_t -- <class 'lv_theme_style_btn_t'>
- theme_style_imgbtn_t -- <class 'lv_theme_style_imgbtn_t'>
- theme_style_label_t -- <class 'lv_theme_style_label_t'>
- theme_style_img_t -- <class 'lv_theme_style_img_t'>
- theme_style_line_t -- <class 'lv_theme_style_line_t'>
- theme_style_bar_t -- <class 'lv_theme_style_bar_t'>
- theme_style_slider_t -- <class 'lv_theme_style_slider_t'>
- theme_style_sw_t -- <class 'lv_theme_style_sw_t'>
- theme_style_cb_t -- <class 'lv_theme_style_cb_t'>
- theme_style_cb_box_t -- <class 'lv_theme_style_cb_box_t'>
- theme_style_btnm_t -- <class 'lv_theme_style_btnm_t'>
- theme_style_btnm_btn_t -- <class 'lv_theme_style_btnm_btn_t'>
- theme_style_mbox_t -- <class 'lv_theme_style_mbox_t'>
- theme_style_mbox_btn_t -- <class 'lv_theme_style_mbox_btn_t'>
- theme_style_page_t -- <class 'lv_theme_style_page_t'>
- theme_style_ta_t -- <class 'lv_theme_style_ta_t'>
- theme_style_spinbox_t -- <class 'lv_theme_style_spinbox_t'>
- theme_style_list_t -- <class 'lv_theme_style_list_t'>
- theme_style_list_btn_t -- <class 'lv_theme_style_list_btn_t'>
- theme_style_ddlist_t -- <class 'lv_theme_style_ddlist_t'>
- theme_style_roller_t -- <class 'lv_theme_style_roller_t'>
- theme_style_tabview_t -- <class 'lv_theme_style_tabview_t'>
- theme_style_tabview_btn_t -- <class 'lv_theme_style_tabview_btn_t'>
- theme_style_tileview_t -- <class 'lv_theme_style_tileview_t'>
- theme_style_table_t -- <class 'lv_theme_style_table_t'>
- theme_style_win_t -- <class 'lv_theme_style_win_t'>
- theme_style_win_btn_t -- <class 'lv_theme_style_win_btn_t'>
- theme_group_t -- <class 'lv_theme_group_t'>
- draw_label_txt_sel_t -- <class 'lv_draw_label_txt_sel_t'>
- draw_label_hint_t -- <class 'lv_draw_label_hint_t'>
- color_t -- <class 'lv_color16_t'>

- `font_code_saver_12` -- struct lv_font_t
- `font_code_saver_16` -- struct lv_font_t
- `font_unscii8_8` -- struct lv_font_t
- `style_scr` -- struct lv_style_t
- `style_transp` -- struct lv_style_t
- `style_transp_fit` -- struct lv_style_t
- `style_transp_tight` -- struct lv_style_t
- `style_plain` -- struct lv_style_t
- `style_plain_color` -- struct lv_style_t
- `style_pretty` -- struct lv_style_t
- `style_pretty_color` -- struct lv_style_t
- `style_btn_rel` -- struct lv_style_t
- `style_btn_pr` -- struct lv_style_t
- `style_btn_tgl_rel` -- struct lv_style_t
- `style_btn_tgl_pr` -- struct lv_style_t
- `style_btn_ina` -- struct lv_style_t
- `_nesting` -- struct _lv_mp_int_wrapper
- `LvReferenceError` -- <class 'LvReferenceError'>

4.3. Zero Micropython + LVGL examples

4.3.1. Importing library and show help

```
import lvgl as lv
help(lv)
```

```
NORMAL FLOAT AUTO REAL RADIAN NAT

  style_transp_fit -- struct lv_style_t
  style_transp_tight -- struct lv_style_t
  style_plain -- struct lv_style_t
  style_plain_color -- struct lv_style_t
  style_pretty -- struct lv_style_t
  style_pretty_color -- struct lv_style_t
  style_btn_rel -- struct lv_style_t
  style_btn_pr -- struct lv_style_t
  style_btn_tgl_rel -- struct lv_style_t
  style_btn_tgl_pr -- struct lv_style_t
  style_btn_ina -- struct lv_style_t
  _nesting -- struct _lv_mp_int_wrapper
  LvReferenceError -- <class 'LvReferenceErro
r'>


>>>
```

4.3.2. Arc

```
import lvgl as lv

# Create style for the Arcs
style = lv.style_t()
style.copy(lv.style_plain)
style.line.color = lv.color_make(0x00, 0xFF, 0xFF) # Arc color
style.line.width = 8                      # Arc width
```

```
# Create an Arc
arc = lv.arc(lv.scr_act(), None)
arc.set_style(lv.arc.STYLE.MAIN, style)   # Use the new style
arc.set_angles(90, 60)
arc.set_size(150, 150)
arc.align(None, lv.ALIGN.CENTER, 0, 0)

lv.show()
print("LVGL drawing finished")
```

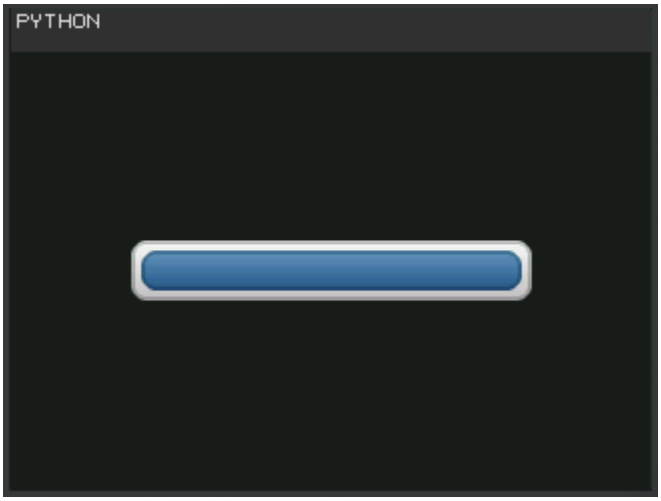
4.3.3. Arc + task

```
import lvgl as lv

# Create an arc which acts as a loader.
class loader_arc(lv.arc):

    def __init__(self, parent, color=lv.color_hex(0x00FF80),
                       width=8, style=lv.style_plain, rate=20):
        super().__init__(parent, None)

        self.a = 0
        self.rate = rate

        # Create style for the Arcs
        self.style = lv.style_t()
        self.style.copy(style)
        self.style.line.color = color
        self.style.line.width = width

        # Create an Arc
        self.set_angles(180, 180)
        self.set_style(self.STYLE.MAIN, self.style)

        # Spin the Arc
        self.spin()

    def spin(self):
        # Create an `lv_task` to update the arc.
        lv.task_create(self.task_cb, self.rate, lv.TASK_PRIO.LOWEST, {})


    # An `lv_task` to call periodically to set the angles of the arc
    def task_cb(self, task):
        self.a+=5
        if self.a >= 359: self.a = 359
```

```
        if self.a < 180: self.set_angles(180-self.a, 180)
        else: self.set_angles(540-self.a, 180)

        if self.a == 359:
            self.a = 0
            task._del()

# Create a loader arc
loader_arc = loader_arc(lv.scr_act())
loader_arc.align(None, lv.ALIGN.CENTER, 0, 0)

lv.show()
print("LVGL drawing finished")
```

4.3.4. Bar

```
import lvgl as lv

bar1 = lv.bar(lv.scr_act(), None)
bar1.set_size(200, 30)
bar1.align(None, lv.ALIGN.CENTER, 0, 0)
bar1.set_anim_time(2500)
bar1.set_value(100, lv.ANIM.ON)

lv.show()
print("LVGL drawing finished")
```



4.3.5. Button matrix

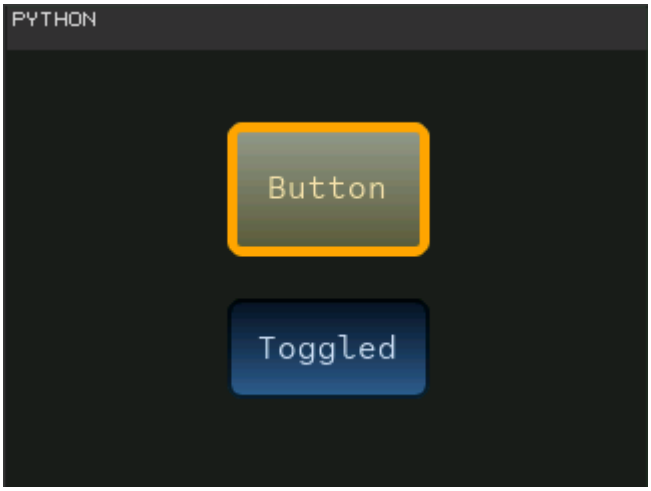
```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
```

```
        txt = obj.get_active_btn_text()
        print("%s was pressed" % txt)

btnm1 = lv.btnm(lv.scr_act(), None)
btnm1.set_map( ["1", "2", "3", "4", "5", "\n",
            "6", "7", "8", "9", "0", "\n",
            "Action1", "Action2", ""])
btnm1.set_btn_width(10, 2)        # Make "Action1" twice as wide as "Action2"
btnm1.align(None, lv.ALIGN.CENTER, 0, 0)
btnm1.set_event_cb(event_handler)

group.add_obj(btnm1)

lv.show()
print("LVGL drawing finished")
```

4.3.6. Buttons

```
 import lvgl as lv

def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        print("Button clicked")

btn1 = lv.btn(lv.scr_act(), None)
btn1.align(None, lv.ALIGN.CENTER, 0, -40)
btn1.set_event_cb(event_handler)
label = lv.label(btn1, None)
label.set_text("Button")

btn2 = lv.btn(lv.scr_act(), None)
# callback can be lambda:
btn2.set_event_cb(lambda obj, event: print("Toggled") if event == lv.EVENT.VALUE_CHANGED else None)
btn2.align(None, lv.ALIGN.CENTER, 0, 40)
btn2.set_toggle(True)
btn2.toggle()
btn2.set_fit2(lv.FIT.NONE, lv.FIT.TIGHT)

label = lv.label(btn2, None)
label.set_text("Toggled")

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()
```

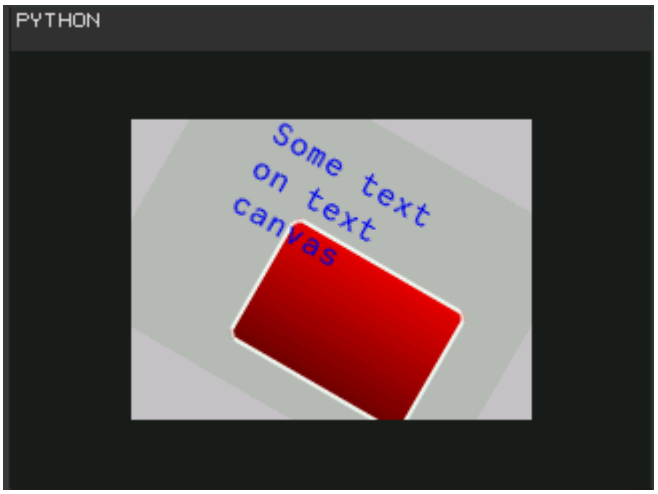
```
# Create groupe for buttons
group = lv.group_create()
group.add_obj(btn1)
group.add_obj(btn2)
keyboard.set_group(group)

lv.show()
print("LVGL drawing finished")
```



4.3.7. Canvas

```
import lvgl as lv

CANVAS_WIDTH  = 200
CANVAS_HEIGHT = 150

style = lv.style_t()
style.copy(lv.style_plain)
style.body.main_color = lv.color_make(0xFF,0,0)
style.body.grad_color = lv.color_make(0x80,0,0)
style.body.radius = 4
style.body.border.width = 2
style.body.border.color = lv.color_make(0xFF,0xFF,0xFF)
style.body.shadow.color = lv.color_make(0xFF,0xFF,0xFF)
style.body.shadow.width = 4
style.line.width = 2
style.line.color = lv.color_make(0,0,0)
style.text.color = lv.color_make(0,0,0xFF)

# CF.TRUE_COLOR requires 4 bytes per pixel
cbuf = bytearray(CANVAS_WIDTH * CANVAS_HEIGHT * 4)

canvas = lv.canvas(lv.scr_act(), None)
canvas.set_buffer(cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, lv.img.CF.TRUE_COLOR)
canvas.align(None, lv.ALIGN.CENTER, 0, 0)
canvas.fill_bg(lv.color_make(0xC0, 0xC0, 0xC0))

canvas.draw_rect(70, 60, 100, 70, style)

canvas.draw_text(40, 20, 100, style, "Some text on text canvas", lv.label.ALIGN.LEFT)

# Test the rotation. It requires an other buffer where the original image is stored.
# So copy the current image to buffer and rotate it to the canvas
img = lv.img_dsc_t()
img.data = cbuf[:]
img.header.cf = lv.img.CF.TRUE_COLOR
```

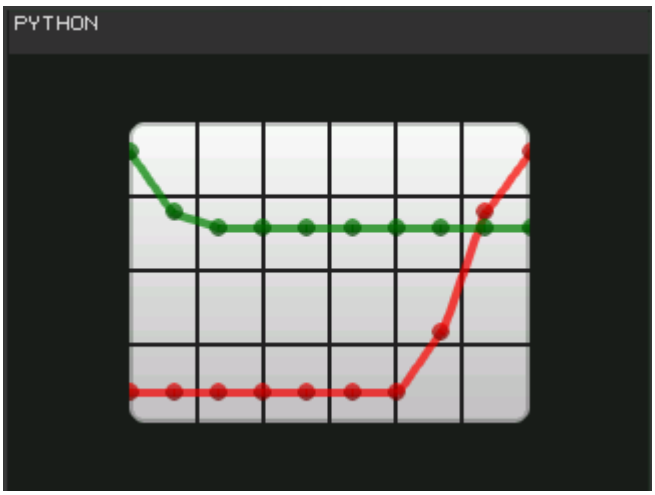
```
img.header.w = CANVAS_WIDTH
img.header.h = CANVAS_HEIGHT

canvas.fill_bg(lv.color_make(0xC0, 0xC0, 0xC0))
canvas.rotate(img, 30, 0, 0, CANVAS_WIDTH // 2, CANVAS_HEIGHT // 2)

lv.show()
```



4.3.8. Chart

```
import lvgl as lv

# Create a chart
chart = lv.chart(lv.scr_act(), None)
chart.set_size(200, 150)
chart.align(None, lv.ALIGN.CENTER, 0, 0)
chart.set_type(lv.chart.TYPE.POINT | lv.chart.TYPE.LINE)   # Show lines and points too
chart.set_series_opa(lv.OPA._70)                           # Opacity of the data series
chart.set_series_width(4)                                  # Line width and point radious

chart.set_range(0, 100)

# Add two data series
ser1 = chart.add_series(lv.color_make(0xFF, 0x00, 0x00))
ser2 = chart.add_series(lv.color_make(0x00, 0x80, 0x00))

# Set points on 'dl1'
chart.set_points(ser1, [10, 10, 10, 10, 10, 10, 10, 30, 70, 90])

# Set points on 'dl2'
chart.set_points(ser2, [90, 70, 65, 65, 65, 65, 65, 65, 65, 65])

lv.show()
print("LVGL drawing finished")
```

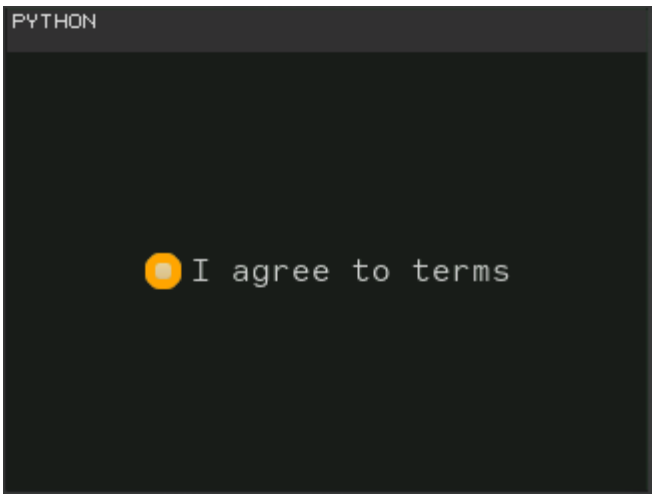
PYTHON



4.3.9. Checkbox

```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("State: %s" % ("Checked" if obj.is_checked() else "Unchecked"))

cb = lv.cb(lv.scr_act(), None)
cb.set_text("I agree to terms")
cb.align(None, lv.ALIGN.CENTER, 0, 0)
cb.set_event_cb(event_handler)

group.add_obj(cb)

lv.show()
print("LVGL drawing finished")
```

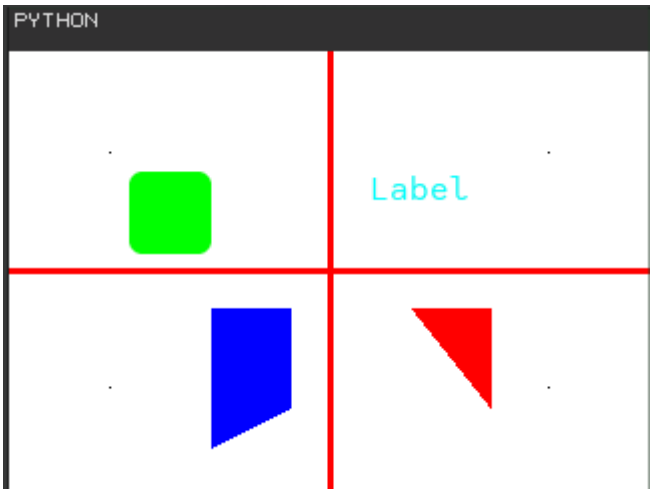
PYTHON



4.3.10. Manual drawing

```
 import lvgl as lv

scr = lv.scr_act()

# Height of persistent header on screen
header_height = lv.get_header_height()

print("Screen width: ", scr.get_width())
print("Screen height: ", scr.get_height())
print("Header height: ", header_height)

def draw_cb(obj, mask, mode):
    if mode == lv.DESIGN.DRAW_MAIN:
        objArea = lv.area_t()
        obj.get_coords(objArea)

        # Fill bg
        bgColor = lv.color_make(0xFF, 0xFF, 0xFF)
        lv.draw_fill(objArea, mask, bgColor, lv.OPA._100)

        styleLine = lv.style_t()
        styleLine.copy(lv.style_plain)
        styleLine.line.color = lv.color_make(0xFF, 0x00, 0x00)
        styleLine.line.width = 3
        styleLine.line.rounded = 1

        # Draw hor line (in coordinates of display (320x240))
        horLinePoint1 = { "x": objArea.x1, "y": objArea.y1 + int(objArea.get_height() / 2) }
        horLinePoint2 = { "x": objArea.x2, "y": objArea.y1 + int(objArea.get_height() / 2) }
        lv.draw_line(horLinePoint1, horLinePoint2, mask, styleLine, lv.OPA._100)

        # Draw ver line (in coordinates of display (320x240))
        verLinePoint1 = { "x": objArea.x1 + int(objArea.get_width() / 2), "y": objArea.y1 }
        verLinePoint2 = { "x": objArea.x1 + int(objArea.get_width() / 2), "y": objArea.y2 }
        lv.draw_line(verLinePoint1, verLinePoint2, mask, styleLine, lv.OPA._100)

        # Draw four pixels
        pxColor = lv.color_make(0x00, 0x00, 0x00)
        lv.draw_px(objArea.x1 + 50, objArea.y1 + 50, mask, pxColor, lv.OPA._100)
        lv.draw_px(objArea.x1 + 50, objArea.y2 - 50, mask, pxColor, lv.OPA._100)
        lv.draw_px(objArea.x2 - 50, objArea.y1 + 50, mask, pxColor, lv.OPA._100)
        lv.draw_px(objArea.x2 - 50, objArea.y2 - 50, mask, pxColor, lv.OPA._100)

        # Draw rectangle
        styleRect = lv.style_t()
        styleRect.copy(lv.style_plain)
        styleRect.body.main_color = lv.color_make(0x00, 0xFF, 0x00)
        styleRect.body.grad_color = lv.color_make(0x00, 0xFF, 0x00)
        styleRect.body.radius = 5

        rectArea = lv.area_t()
        rectArea.set(objArea.x1 + 60, objArea.y1 + 60, objArea.x1 + 100, objArea.y1 + 100)
        lv.draw_rect(rectArea, mask, styleRect, lv.OPA._100)

        # Draw label
        labelStyle = lv.style_t()
        labelStyle.copy(lv.style_plain)
        labelStyle.text.color = lv.color_make(0x00, 0xFF, 0xFF)

        labelOffset = {"x": 0, "y": 0}

        labelArea = lv.area_t()
        labelArea.set(objArea.x1 + 180, objArea.y1 + 60, objArea.x2, objArea.y1 + 80)
        lv.draw_label(labelArea, mask, labelStyle, lv.OPA._100, "Label", 0, labelOffset, None, None, l
```

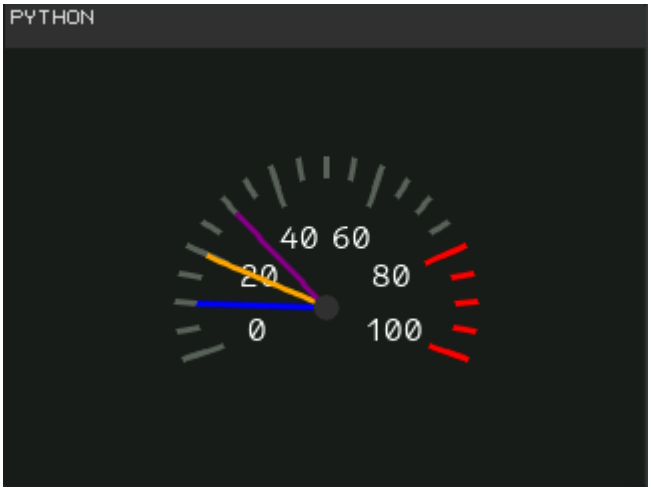
```
v.BIDI_DIR.LTR)

        # Draw triangle
        styleRect.body.main_color = lv.color_make(0xFF, 0x00, 0x00)
        trigPoints = [ {"x":200, "y":150},
                       {"x":240, "y":150},
                       {"x":240, "y":200}]
        lv.draw_triangle(trigPoints, mask, styleRect, lv.OPA._100)

        # Draw polygon
        styleRect.body.main_color = lv.color_make(0x00, 0x00, 0xFF)
        polyPoints = [ {"x":100, "y":150},
                       {"x":140, "y":150},
                       {"x":140, "y":200},
                       {"x":100, "y":220}]
        lv.draw_polygon(polyPoints, len(polyPoints), mask, styleRect, lv.OPA._100)

        return True
    else:
        return False

scr.set_design_cb(draw_cb)

print("Main draw finished")

lv.show()
```



4.3.11. Gauge

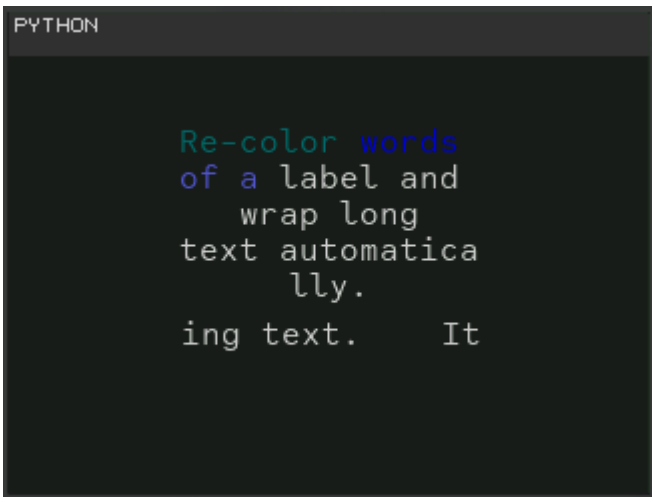
```
 import lvgl as lv

# Create a style
style = lv.style_t()
style.copy(lv.style_pretty_color)
style.body.main_color = lv.color_hex3(0x666)     # Line color at the beginning
style.body.grad_color =  lv.color_hex3(0x666)    # Line color at the end
style.body.padding.left = 10                     # Scale line length
style.body.padding.inner = 8                     # Scale label padding
style.body.border.color = lv.color_hex3(0x333)   # Needle middle circle color
style.line.width = 3
style.text.color = lv.color_hex3(0xFFFFFF)
style.line.color = lv.color_hex3(0xF00)          # Line color after the critical value

# Describe the color for the needles
needle_colors = [
    lv.color_make(0x00, 0x00, 0xFF),
    lv.color_make(0xFF, 0xA5, 0x00),
```

```
    lv.color_make(0x80, 0x00, 0x80)
]

# Create a gauge
gauge1 = lv.gauge(lv.scr_act(), None)
gauge1.set_style(lv.gauge.STYLE.MAIN, style)
gauge1.set_needle_count(len(needle_colors), needle_colors)
gauge1.set_size(150, 150)
gauge1.align(None, lv.ALIGN.CENTER, 0, 20)

# Set the values
gauge1.set_value(0, 10)
gauge1.set_value(1, 20)
gauge1.set_value(2, 30)

lv.show()
print("LVGL drawing finished")
```



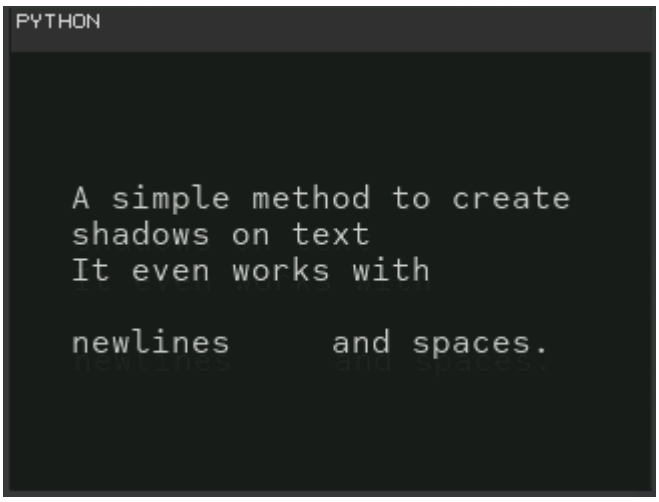
4.3.12. Label

```
import lvgl as lv

label1 = lv.label(lv.scr_act(), None)
label1.set_long_mode(lv.label.LONG.BREAK)     # Break the long lines
label1.set_recolor(True)                      # Enable re-coloring by commands in the text
label1.set_align(lv.label.ALIGN.CENTER)       # Center aligned lines
# Use special symbol \377 to set text color
label1.set_text("\377008080 Re-color\377 \3770000ff words\377 \3776666ff of a\377 label " +
                "and  wrap long text automatically.")
label1.set_width(150)
label1.align(None, lv.ALIGN.CENTER, 0, -30)

label2 = lv.label(lv.scr_act(), None)
label2.set_long_mode(lv.label.LONG.SROLL_CIRC)     # Circular scroll
label2.set_width(150)
label2.set_text("It is a circularly scrolling text. ")
label2.align(None, lv.ALIGN.CENTER, 0, 30)

lv.show()
print("LVGL drawing finished")
```



4.3.13. Label shadow

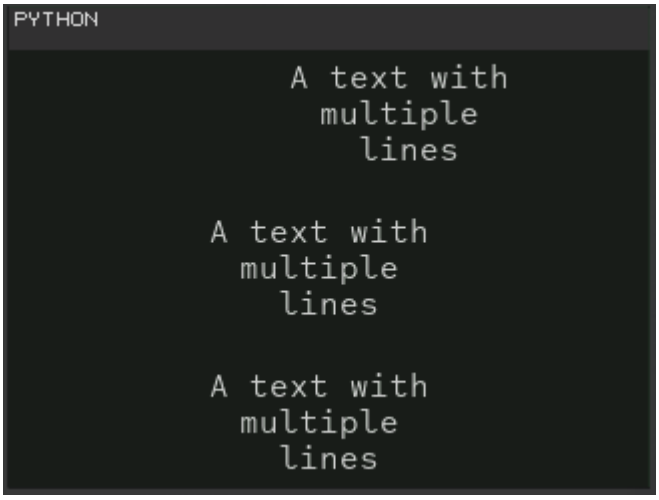
```python
import lvgl as lv

# Create a style for the shadow
label_style = lv.style_t()
label_style.copy(lv.style_plain)
label_style.text.opa = lv.OPA._50

# Create a label for the shadow first (it's in the background)
shadow_label = lv.label(lv.scr_act(), None)
shadow_label.set_style(lv.label.STYLE.MAIN, label_style)

# Create the main label
main_label = lv.label(lv.scr_act(), None)
main_label.set_text("A simple method to create\n" +
                            "shadows on text\n" +
                            "It even works with\n\n" +
                            "newlines     and spaces.")

# Set the same text for the shadow label
shadow_label.set_text(main_label.get_text())

# Position the main label
main_label.align(None, lv.ALIGN.CENTER, 0, 0)

# Shift the second label down and to the right by 1 pixel
shadow_label.align(main_label, lv.ALIGN.IN_TOP_LEFT, 1, 1)

lv.show()
print("LVGL drawing finished")
```



4.3.14. Label change

```python
import lvgl as lv

# Create three labels to demonstrate the alignments.
labels = []

# `lv_label_set_align` is not required to align the object itslef.
# It's used only when the text has multiple lines

# Create a label on the top.
# No additional alignment so it will be the reference
label = lv.label(lv.scr_act(), None)
label.align(None, lv.ALIGN.IN_TOP_MID, 0, 5)
label.set_align(lv.label.ALIGN.CENTER)
labels.append(label)

# Create a label in the middle.
# `lv_obj_align` will be called every time the text changes
# to keep the middle position
label = lv.label(lv.scr_act(), None)
label.align(None, lv.ALIGN.CENTER, 0, 0)
label.set_align(lv.label.ALIGN.CENTER)
labels.append(label)

# Create a label in the bottom.
# Enable auto realign.
label = lv.label(lv.scr_act(), None)
label.set_auto_realign(True)
label.align(None, lv.ALIGN.IN_BOTTOM_MID, 0, -5)
label.set_align(lv.label.ALIGN.CENTER)
labels.append(label)

class TextChanger:
    """Changes texts of all labels every second"""
    def __init__(self, labels,
                 texts=["Text", "A very long text", "A text with\nmultiple\nlines"],
                 rate=1000):
        self.texts = texts
        self.labels = labels
        self.rate = rate
        self.counter = 0

    def start(self):
        lv.task_create(self.task_cb, self.rate, lv.TASK_PRIO.LOWEST, None)

    def task_cb(self, task):
```

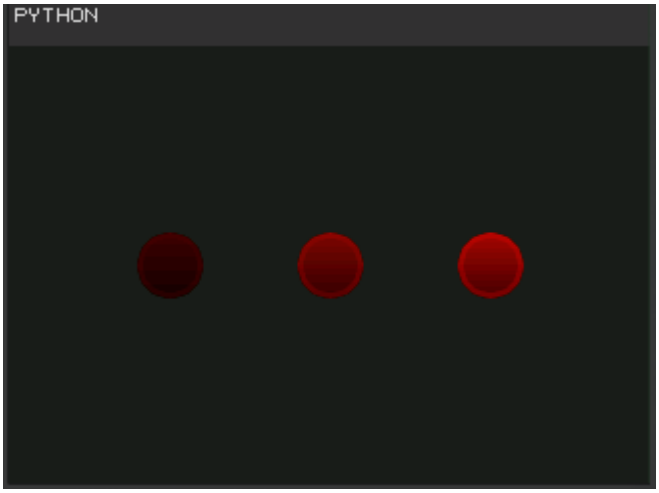
```
        for label in labels:
            label.set_text(self.texts[self.counter])

        # Manually realaign `labels[1]`
        if len(self.labels) > 1:
            self.labels[1].align(None, lv.ALIGN.CENTER, 0, 0)

        self.counter = self.counter + 1
        print("Counter: ", self.counter)
        if self.counter >= len(self.texts):
            task._del()
            print("Task deleted")

text_changer = TextChanger(labels)
text_changer.start()

lv.show()
print("LVGL drawing finished")
```



4.3.15. Led

```
import lvgl as lv

# Create a style for the LED
style_led = lv.style_t()
style_led.copy(lv.style_pretty_color)
style_led.body.radius = 800 # large enough to draw a circle
style_led.body.main_color = lv.color_make(0xb5, 0x0f, 0x04)
style_led.body.grad_color = lv.color_make(0x50, 0x07, 0x02)
style_led.body.border.color = lv.color_make(0xfa, 0x0f, 0x00)
style_led.body.border.width = 3
style_led.body.border.opa = lv.OPA._30
style_led.body.shadow.color = lv.color_make(0xb5, 0x0f, 0x04)
style_led.body.shadow.width = 5

# Create a LED and switch it OFF
led1 = lv.led(lv.scr_act(), None)
led1.set_style(lv.led.STYLE.MAIN, style_led)
led1.align(None, lv.ALIGN.CENTER, -80, 0)
led1.off()

# Copy the previous LED and set a brightness
led2 = lv.led(lv.scr_act(), led1)
led2.align(None, lv.ALIGN.CENTER, 0, 0)
led2.set_bright(190)

# Copy the previous LED and switch it ON
```

```
led3 = lv.led(lv.scr_act(), led1)
led3.align(None, lv.ALIGN.CENTER, 80, 0)
led3.on()

lv.show()
```

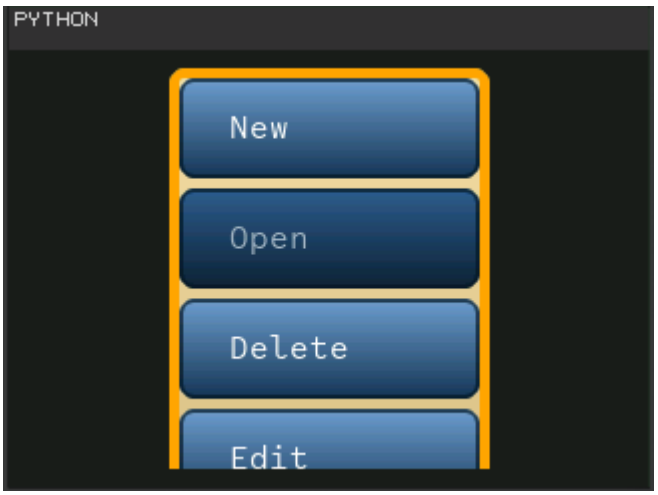


4.3.16. Line

```
import lvgl as lv

# Create an array for the points of the line
line_points = [ {"x":5, "y":5},
                {"x":70, "y":70},
                {"x":120, "y":10},
                {"x":180, "y":60},
                {"x":240, "y":10}]

# Create new style (thick dark blue)
style_line = lv.style_t()
style_line.copy(lv.style_plain)
style_line.line.color = lv.color_make(0x00, 0x3b, 0x75)
style_line.line.width = 3
style_line.line.rounded = 1

# Copy the previous line and apply the new style
line1 = lv.line(lv.scr_act(), None)
line1.set_points(line_points, len(line_points))      # Set the points
line1.set_style(lv.line.STYLE.MAIN, style_line)
line1.align(None, lv.ALIGN.CENTER, 0, 0)

lv.show()
```

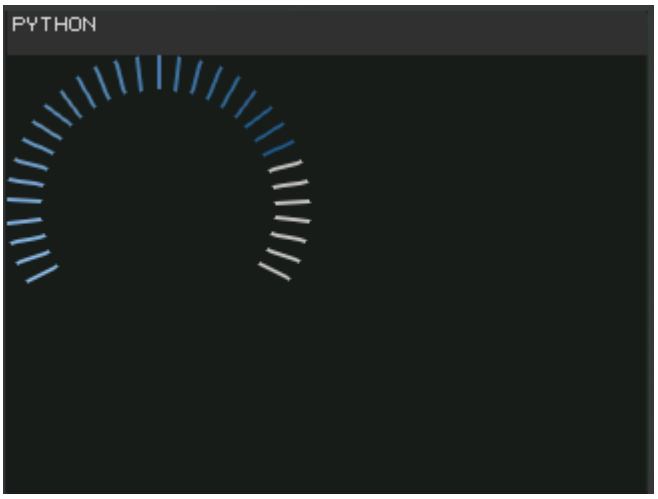
4.3.17. List

```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

# Create a list
list1 = lv.list(lv.scr_act(), None)
list1.set_size(160, 200)
list1.align(None, lv.ALIGN.CENTER, 0, 0)

# Add buttons to the list
def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        print("Clicked: %s" % list1.get_btn_text(obj))

list_btn = list1.add_btn(None, "New")
list_btn.set_event_cb(event_handler)

list_btn = list1.add_btn(None, "Open")
list_btn.set_event_cb(event_handler)

list_btn = list1.add_btn(None, "Delete")
list_btn.set_event_cb(event_handler)

list_btn = list1.add_btn(None, "Edit")
list_btn.set_event_cb(event_handler)

list_btn = list1.add_btn(None, "Save")
list_btn.set_event_cb(event_handler)

group.add_obj(list1)

lv.show()
```



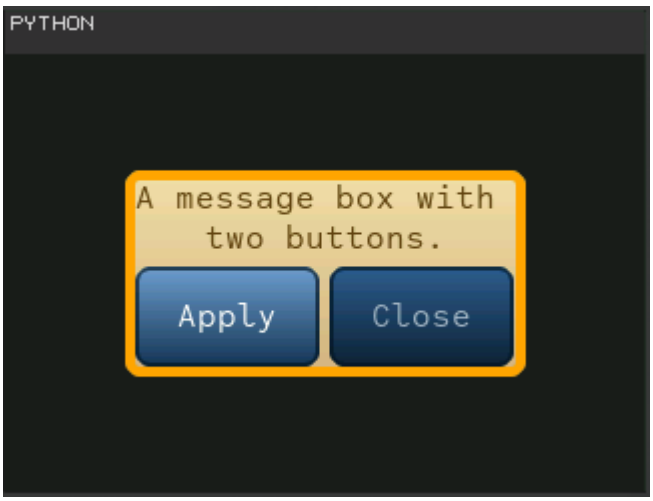
4.3.18. LMeter

```
import lvgl as lv

# Create a style for the line meter
style_lmeter = lv.style_t()
style_lmeter.copy(lv.style_pretty_color)
style_lmeter.line.width = 2
style_lmeter.line.color = lv.color_hex(0xc0c0c0)              # Silver
style_lmeter.body.main_color = lv.color_hex(0x91bfed)         # Light blue
style_lmeter.body.grad_color = lv.color_hex(0x04386c)         # Dark blue
style_lmeter.body.padding.left = 16                           # Line length

# Create a line meter
lmeter = lv.lmeter(lv.scr_act(), None)
lmeter.set_range(0, 100)                    # Set the range
lmeter.set_value(80)                        # Set the current value
lmeter.set_scale(240, 31)                   # Set the angle and number of lines
lmeter.set_style(lv.lmeter.STYLE.MAIN, style_lmeter)          # Apply the new style
lmeter.set_size(150, 150)

lv.show()
```



4.3.19. MessageBox

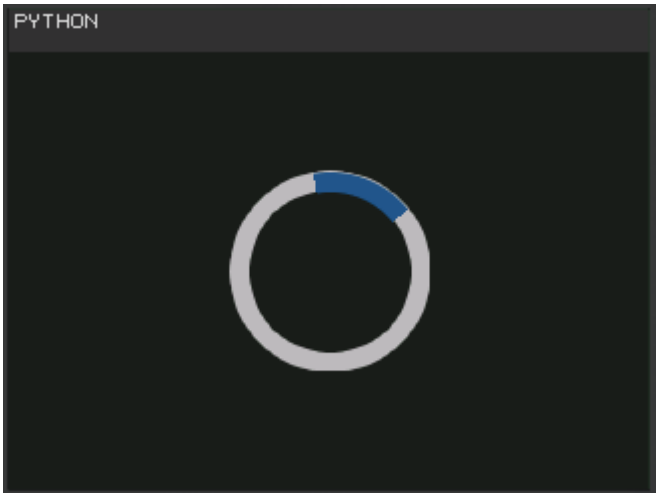
```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("Button: %s" % lv.mbox.get_active_btn_text(obj))

mbox1 = lv.mbox(lv.scr_act(), None)
mbox1.set_text("A message box with two buttons.")
mbox1.add_btns(["Apply", "Close", ""])
mbox1.set_width(200)
mbox1.set_event_cb(event_handler)
mbox1.align(None, lv.ALIGN.CENTER, 0, 0)  # Align to the corner

group.add_obj(mbox1)

lv.show()
```



4.3.20. Preloader

```
import lvgl as lv

# Create a style for the Preloader
style = lv.style_t()
style.copy(lv.style_plain)
style.line.width = 10                       # 10 px thick arc
style.line.color = lv.color_hex3(0x258)     # Blueish arc color

style.body.border.color = lv.color_hex3(0xBBB) # Gray background color
style.body.border.width = 10
style.body.padding.left = 0

# Create a Preloader object
preload = lv.preload(lv.scr_act(), None)
```

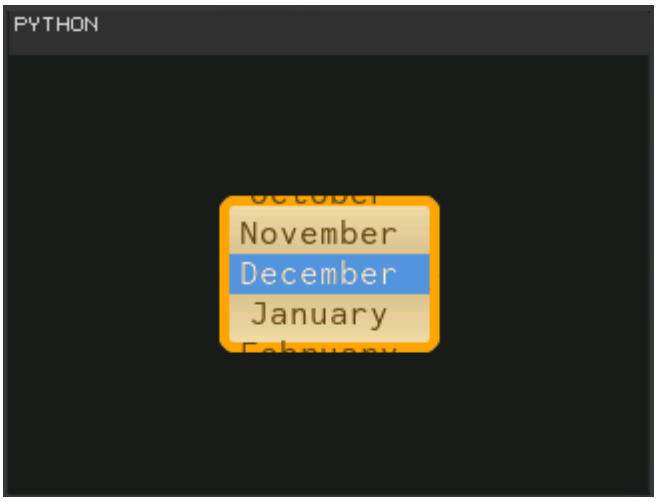
```
preload.set_size(100, 100)
preload.align(None, lv.ALIGN.CENTER, 0, 0)
preload.set_style(lv.preload.STYLE.MAIN, style)

lv.show()
```



4.3.21. Roller

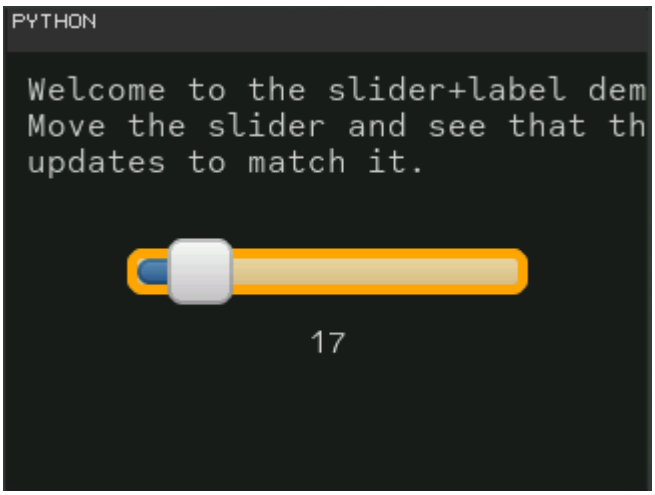
```
 import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        option = " "*10
        obj.get_selected_str(option, len(option))
        print("Selected month: %s" % option.strip())

roller1 = lv.roller(lv.scr_act(), None)
roller1.set_options("\n".join([
                    "January",
                    "February",
                    "March",
                    "April",
                    "May",
                    "June",
                    "July",
                    "August",
                    "September",
                    "October",
                    "November",
                    "December"]), lv.roller.MODE.INIFINITE)

roller1.set_visible_row_count(4)
roller1.align(None, lv.ALIGN.CENTER, 0, 0)
roller1.set_event_cb(event_handler)
```

```
group.add_obj(roller1)

lv.show()
```



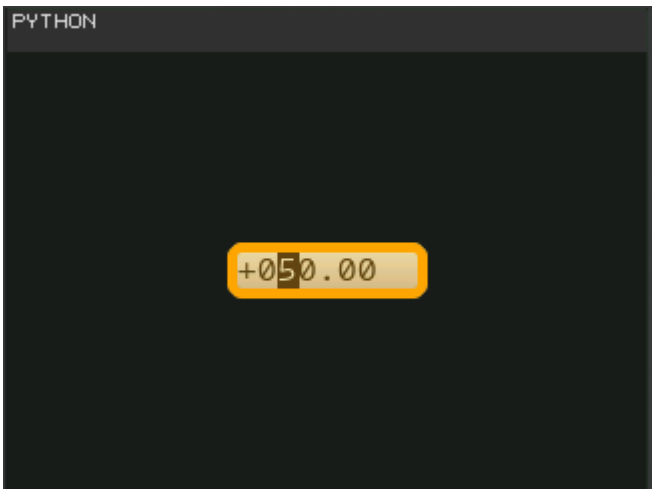
4.3.22. Slider

```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

# Create a label below the slider
slider_label = lv.label(lv.scr_act(), None)
slider_label.set_text("0")
slider_label.set_auto_realign(True)

def slider_event_cb(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        slider_label.set_text("%u" % obj.get_value())

# Create a slider in the center of the display
slider = lv.slider(lv.scr_act(), None)
slider.set_width(200)
slider.align(None, lv.ALIGN.CENTER, 0, 0)
slider.set_event_cb(slider_event_cb)
slider.set_range(0, 100)

slider_label.align(slider, lv.ALIGN.OUT_BOTTOM_MID, 0, 10)

# Create an informative label
info = lv.label(lv.scr_act(), None)
info.set_text("""Welcome to the slider+label demo!
Move the slider and see that the label
updates to match it.""")
info.align(None, lv.ALIGN.IN_TOP_LEFT, 10, 10)

group.add_obj(slider)

lv.show()
```



4.3.23. Spinbox

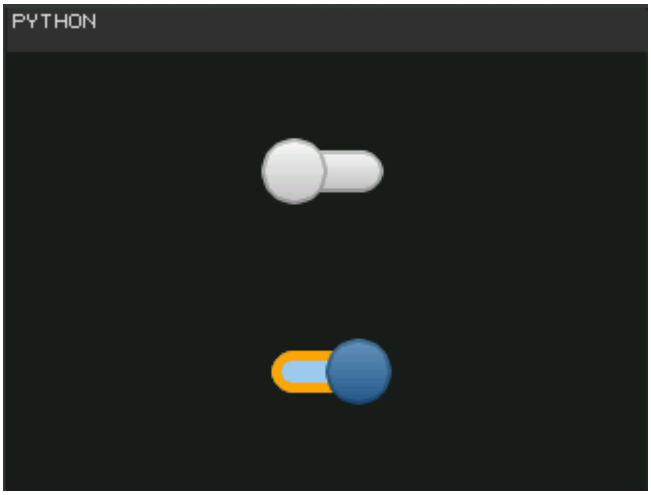
```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("Value: %d" % obj.get_value())
    elif event == lv.EVENT.CLICKED:
        # For simple test: Click the spinbox to increment its value
        obj.increment()

spinbox = lv.spinbox(lv.scr_act(), None)
spinbox.set_digit_format(5, 3)
spinbox.step_prev()
spinbox.set_width(100)
spinbox.align(None, lv.ALIGN.CENTER, 0, 0)
spinbox.set_event_cb(event_handler)

group.add_obj(spinbox)

lv.show()
```



4.3.24. Switch

```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("State: %s" % ("On" if obj.get_state() else "Off"))

# Create styles for the switch
bg_style = lv.style_t()
indic_style = lv.style_t()
knob_on_style = lv.style_t()
knob_off_style = lv.style_t()

bg_style.copy(lv.style_pretty)
bg_style.body.radius = 800
bg_style.body.padding.top = 6
bg_style.body.padding.bottom = 6

indic_style.copy(lv.style_pretty_color)
indic_style.body.radius = 800
indic_style.body.main_color = lv.color_hex(0x9fc8ef)
indic_style.body.grad_color = lv.color_hex(0x9fc8ef)
indic_style.body.padding.left = 0
indic_style.body.padding.right = 0
indic_style.body.padding.top = 0
indic_style.body.padding.bottom = 0

knob_off_style.copy(lv.style_pretty)
knob_off_style.body.radius = 800
knob_off_style.body.shadow.width = 4
knob_off_style.body.shadow.type = lv.SHADOW.BOTTOM

knob_on_style.copy(lv.style_pretty_color)
knob_on_style.body.radius = 800
knob_on_style.body.shadow.width = 4
```

```
knob_on_style.body.shadow.type = lv.SHADOW.BOTTOM

# Create a switch and apply the styles
sw1 = lv.sw(lv.scr_act(), None)
sw1.set_style(lv.sw.STYLE.BG, bg_style)
sw1.set_style(lv.sw.STYLE.INDIC, indic_style)
sw1.set_style(lv.sw.STYLE.KNOB_ON, knob_on_style)
sw1.set_style(lv.sw.STYLE.KNOB_OFF, knob_off_style)
sw1.align(None, lv.ALIGN.CENTER, 0, -50)
sw1.set_event_cb(event_handler)

# Copy the first switch and turn it ON
sw2 = lv.sw(lv.scr_act(), None)
sw2.set_style(lv.sw.STYLE.BG, bg_style)
sw2.set_style(lv.sw.STYLE.INDIC, indic_style)
sw2.set_style(lv.sw.STYLE.KNOB_ON, knob_on_style)
sw2.set_style(lv.sw.STYLE.KNOB_OFF, knob_off_style)
sw2.on(lv.ANIM.ON)
sw2.align(None, lv.ALIGN.CENTER, 0, 50)
sw2.set_event_cb(lambda o,e: None)

group.add_obj(sw1)
group.add_obj(sw2)

lv.show()
```



4.3.25. Table

```
 import lvgl as lv

# Create a normal cell style
style_cell1 = lv.style_t()
style_cell1.copy(lv.style_plain)
style_cell1.body.border.width = 1
style_cell1.body.border.color = lv.color_make(0,0,0)

# Crealte a header cell style
style_cell2 = lv.style_t()
style_cell2.copy(lv.style_plain)
style_cell2.body.border.width = 1
style_cell2.body.border.color = lv.color_make(0,0,0)
style_cell2.body.main_color = lv.color_make(0xC0, 0xC0, 0xC0)
style_cell2.body.grad_color = lv.color_make(0xC0, 0xC0, 0xC0)

table = lv.table(lv.scr_act(), None)
table.set_style(lv.table.STYLE.CELL1, style_cell1)
table.set_style(lv.table.STYLE.CELL2, style_cell2)
```

```
table.set_style(lv.table.STYLE.BG, lv.style_transp_tight)
table.set_col_cnt(2)
table.set_row_cnt(4)
table.align(None, lv.ALIGN.CENTER, 0, 0)

# Make the cells of the first row center aligned
table.set_cell_align(0, 0, lv.label.ALIGN.CENTER)
table.set_cell_align(0, 1, lv.label.ALIGN.CENTER)

# Make the cells of the first row TYPE = 2 (use `style_cell2`)
table.set_cell_type(0, 0, 2)
table.set_cell_type(0, 1, 2)

# Fill the first column
table.set_cell_value(0, 0, "Name")
table.set_cell_value(1, 0, "Apple")
table.set_cell_value(2, 0, "Banana")
table.set_cell_value(3, 0, "Citron")

# Fill the second column
table.set_cell_value(0, 1, "Price")
table.set_cell_value(1, 1, "$7")
table.set_cell_value(2, 1, "$4")
table.set_cell_value(3, 1, "$6")

lv.show()
```

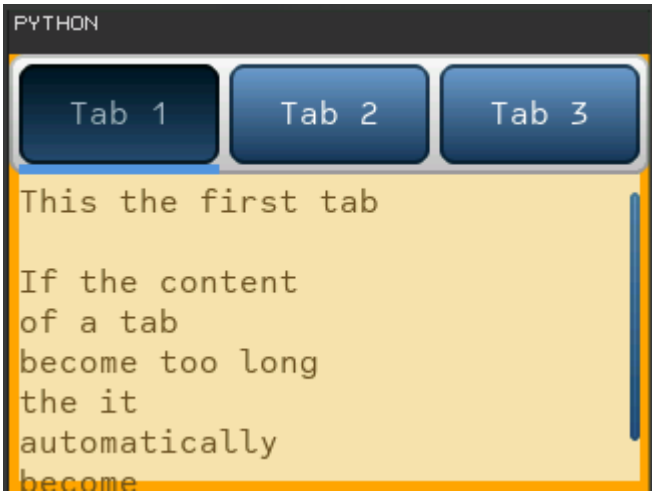
PYTHON

Name	Price
Apple	$7
Banana	$4
Citron	$6

4.3.26. Tabview

```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

# Create a Tab view object
tabview = lv.tabview(lv.scr_act(), None)

# Add 3 tabs (the tabs are page (lv_page) and can be scrolled
tab1 = tabview.add_tab("Tab 1")
```

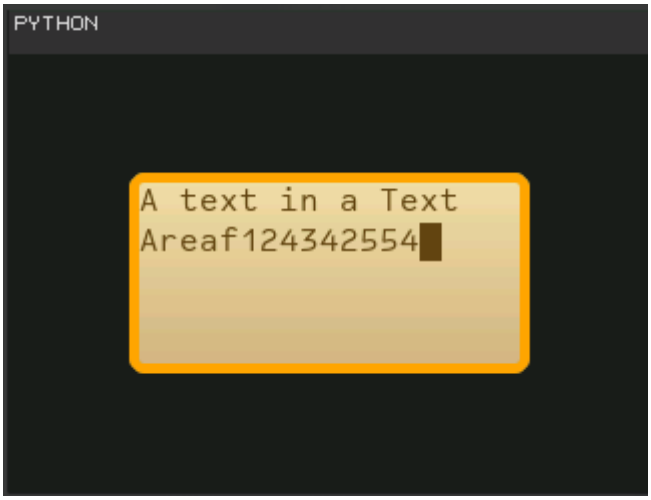
```
tab2 = tabview.add_tab("Tab 2")
tab3 = tabview.add_tab("Tab 3")

# Add content to the tabs
label = lv.label(tab1, None)
label.set_text("""This the first tab

If the content
of a tab
become too long
the it
automatically
become
scrollable.""")

label = lv.label(tab2, None)
label.set_text("Second tab")

label = lv.label(tab3, None)
label.set_text("Third tab")

group.add_obj(tabview)

lv.show()
```



4.3.27. Textarea simple

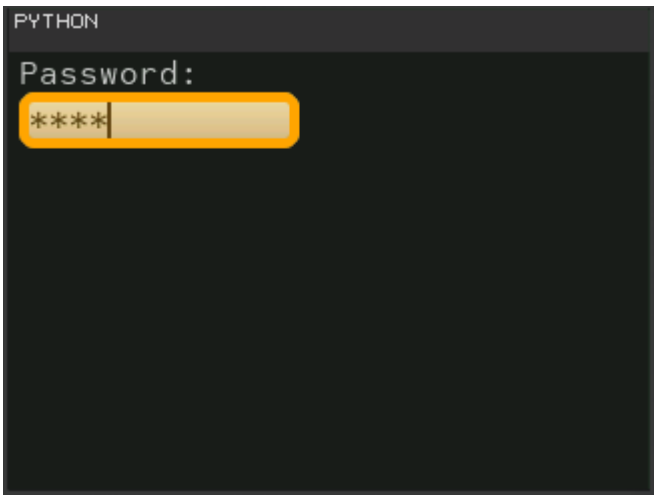
```
import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("Value: %s" % obj.get_text())

ta1 = lv.ta(lv.scr_act(), None)
ta1.set_size(200, 100)
ta1.align(None, lv.ALIGN.CENTER, 0, 0)
```

```
ta1.set_cursor_type(lv.CURSOR.BLOCK)
ta1.set_text("A text in a Text Area")     # Set an initial text
ta1.set_event_cb(event_handler)

group.add_obj(ta1)

lv.show()
print("LVGL drawing finished")
```



4.3.28. Textarea password

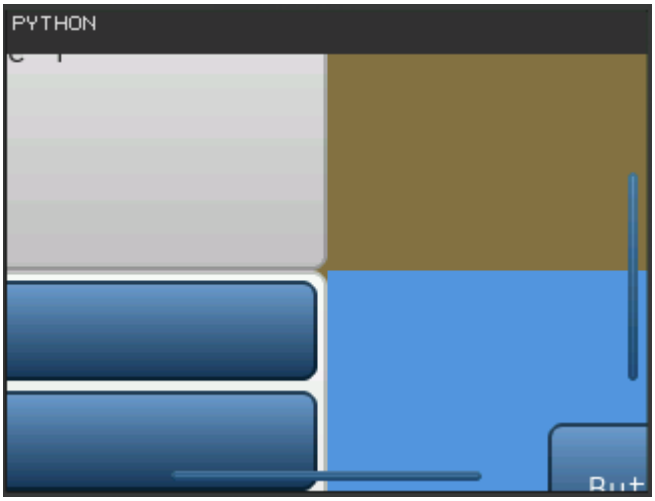
```
 import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

HOR_RES = lv.scr_act().get_width()
print("HOR_RES = %d" % HOR_RES)

# Create the password box
pwd_ta = lv.ta(lv.scr_act(), None)
pwd_ta.set_text("")
pwd_ta.set_pwd_mode(True)
pwd_ta.set_one_line(True)
pwd_ta.set_width(HOR_RES // 2 - 20)
pwd_ta.set_pos(5, 20)

# Create a label and position it above the text box
pwd_label = lv.label(lv.scr_act(), None)
pwd_label.set_text("Password:")
pwd_label.align(pwd_ta, lv.ALIGN.OUT_TOP_LEFT, 0, 0)

group.add_obj(pwd_ta)

lv.show()
print("LVGL drawing finished")
```



4.3.29. Tileview

```
 import lvgl as lv

# Keyboard register
keyboardDriver = lv.indev_drv_t()
keyboardDriver.init()
keyboardDriver.type = lv.INDEV_TYPE.KEYPAD
keyboardDriver.read_cb = lv.keypad_read
keyboard = keyboardDriver.register()

# Create groupe for keyboard handled objects
group = lv.group_create()
keyboard.set_group(group)

valid_pos = [{"x":0, "y": 0}, {"x": 0, "y": 1}, {"x": 1,"y": 1}]

# resolution of the screen
HOR_RES = lv.scr_act().get_width()
VER_RES = lv.scr_act().get_height()

tileview = lv.tileview(lv.scr_act(), None)
tileview.set_valid_positions(valid_pos, len(valid_pos))
tileview.set_edge_flash(True)

# Tile1: just a label
tile1 = lv.obj(tileview, None)
tile1.set_size(HOR_RES, VER_RES)
tile1.set_style(lv.style_pretty)
tileview.add_element(tile1)

label = lv.label(tile1, None)
label.set_text("Tile 1")
label.align(None, lv.ALIGN.CENTER, 0, 0)

# Tile2: a list
lst = lv.list(tileview, None)
lst.set_size(HOR_RES, VER_RES)
lst.set_pos(0, VER_RES)
lst.set_scroll_propagation(True)
lst.set_sb_mode(lv.SB_MODE.OFF)
tileview.add_element(lst)

list_btn = lst.add_btn(None, "One")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Two")
```

```
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Three")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Four")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Five")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Six")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Seven")
tileview.add_element(list_btn)

list_btn = lst.add_btn(None, "Eight")
tileview.add_element(list_btn)

# Tile3: a button
tile3 = lv.obj(tileview, None)
tile3.set_size(HOR_RES, VER_RES)
tile3.set_pos(HOR_RES, VER_RES)
tileview.add_element(tile3)

btn = lv.btn(tile3, None)
btn.align(None, lv.ALIGN.CENTER, 0, 0)

label = lv.label(btn, None)
label.set_text("Button")

group.add_obj(tileview)

lv.show()
```



V. Documentation changelog

v2.27.1 (2025-12-29)

- Added lvgl micropython module objects overview.

v2.27.0 (2025-12-15)

- Enabled functions attributes and enumerate build options.
- Fixed deleting and closing (leak) objects of opened files.
- Fixed memory leaks in functions of os module (chdir, mkdir, remove, unlink, rename, rmdir, stat, ilistdir, open) in case of OSError generation.
- Enabled builtin function reversed(.
- Enabled builtin function help(.

v2.26.0 (2025-10-16)

- Added changelog documentation generating.
- Change styling.
- Added table of content.
- Added f-strings.

v2.25.0 (2025-09-11)

- Fix typo mistakes.

---

1. `https://www.micropython.org/` ↩
2. `https://docs.micropython.org/en/latest/` ↩↩
3. `https://github.com/micropython/micropython` ↩
4. `https://docs.micropython.org/en/latest/genrst/index.html` ↩↩
5. `https://github.com/micropython/micropython/blob/master/py/mpconfig.h` ↩
6. `https://docs.micropython.org/en/latest/library/sys.html` ↩
7. `https://docs.micropython.org/en/latest/library/os.html` ↩
8. `https://docs.micropython.org/en/latest/library/errno.html` ↩
9. `https://docs.micropython.org/en/latest/library/micropython.html` ↩
10. `https://docs.python.org/3/index.html` ↩
11. `https://docs.micropython.org/en/latest/library/index.html` ↩
12. `https://docs.lvgl.io/6.1` ↩
13. `https://docs.lvgl.io/6.1/overview/indev.html#keypad-and-encoder` ↩