

---

# **python-tcod Documentation**

*Release 11.3.0*

**Kyle Stewart**

**Sep 19, 2019**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Windows . . . . .	3
1.2	MacOS . . . . .	3
1.3	Linux (Debian-based) . . . . .	4
1.4	Upgrading python-tcod . . . . .	4
1.5	Upgrading from libtcodpy to python-tcod . . . . .	4
1.6	Distributing . . . . .	4
1.7	Python 2.7 . . . . .	5
<b>2</b>	<b>Glossary</b>	<b>7</b>
<b>3</b>	<b>Changelog</b>	<b>9</b>
3.1	Unreleased . . . . .	9
3.2	11.3.0 - 2019-09-06 . . . . .	9
3.3	11.2.2 - 2019-08-25 . . . . .	9
3.4	11.2.1 - 2019-08-25 . . . . .	9
3.5	11.2.0 - 2019-08-24 . . . . .	9
3.6	11.1.2 - 2019-08-02 . . . . .	10
3.7	11.1.1 - 2019-08-01 . . . . .	10
3.8	11.1.0 - 2019-07-05 . . . . .	10
3.9	11.0.2 - 2019-06-21 . . . . .	10
3.10	11.0.1 - 2019-06-21 . . . . .	11
3.11	11.0.0 - 2019-06-14 . . . . .	11
3.12	10.1.1 - 2019-06-02 . . . . .	11
3.13	10.1.0 - 2019-05-24 . . . . .	11
3.14	10.0.5 - 2019-05-17 . . . . .	11
3.15	10.0.4 - 2019-05-17 . . . . .	11
3.16	10.0.3 - 2019-05-10 . . . . .	12
3.17	10.0.2 - 2019-04-26 . . . . .	12
3.18	10.0.1 - 2019-04-19 . . . . .	12
3.19	10.0.0 - 2019-03-29 . . . . .	12
3.20	9.3.0 - 2019-03-15 . . . . .	12
3.21	9.2.5 - 2019-03-04 . . . . .	13
3.22	9.2.4 - 2019-03-02 . . . . .	13
3.23	9.2.3 - 2019-03-01 . . . . .	13
3.24	9.2.2 - 2019-02-26 . . . . .	13
3.25	9.2.1 - 2019-02-25 . . . . .	13

3.26	9.2.0 - 2019-02-24	14
3.27	9.1.0 - 2019-02-23	14
3.28	9.0.0 - 2019-02-17	14
3.29	8.5.0 - 2019-02-15	14
3.30	8.4.3 - 2019-02-06	15
3.31	8.4.2 - 2019-02-05	15
3.32	8.4.1 - 2019-02-01	15
3.33	8.4.0 - 2019-01-31	15
3.34	8.3.2 - 2018-12-28	16
3.35	8.3.1 - 2018-12-28	16
3.36	8.3.0 - 2018-12-08	16
3.37	8.2.0 - 2018-11-27	16
3.38	8.1.1 - 2018-11-16	17
3.39	8.1.0 - 2018-11-15	17
3.40	8.0.0 - 2018-11-02	17
3.41	7.0.1 - 2018-10-27	17
3.42	7.0.0 - 2018-10-25	17
3.43	6.0.7 - 2018-10-24	18
3.44	6.0.6 - 2018-10-01	18
3.45	6.0.5 - 2018-09-28	18
3.46	6.0.4 - 2018-09-11	18
3.47	6.0.3 - 2018-09-05	18
3.48	6.0.2 - 2018-08-28	18
3.49	6.0.1 - 2018-08-24	18
3.50	6.0.0 - 2018-08-23	19
3.51	5.0.1 - 2018-07-08	19
3.52	5.0.0 - 2018-07-05	19
3.53	4.6.1 - 2018-06-30	19
3.54	4.5.2 - 2018-06-29	19
3.55	4.5.1 - 2018-06-23	20
3.56	4.5.0 - 2018-06-12	20
3.57	4.4.0 - 2018-05-02	20
3.58	4.3.2 - 2018-03-18	20
3.59	4.3.1 - 2018-03-07	21
3.60	4.3.0 - 2018-02-01	21
3.61	4.2.3 - 2018-01-06	21
3.62	4.2.2 - 2018-01-06	21
3.63	4.2.0 - 2018-01-02	21
3.64	4.1.1 - 2017-11-02	22
3.65	4.1.0 - 2017-07-19	22
3.66	4.0.1 - 2017-07-12	22
3.67	4.0.0 - 2017-07-08	22
3.68	3.2.0 - 2017-07-04	23
3.69	3.1.0 - 2017-05-28	23
3.70	3.0.2 - 2017-04-13	23
3.71	3.0.1 - 2017-03-22	23
3.72	3.0.0 - 2017-03-21	24
3.73	2.0.1 - 2017-02-22	24
3.74	2.0.0 - 2017-02-15	24
3.75	1.6.0 - 2016-11-18	24
3.76	1.5.3 - 2016-06-04	24
3.77	1.5.2 - 2016-03-11	25
3.78	1.5.1 - 2015-12-20	25
3.79	1.5.0 - 2015-07-13	25

3.80	1.4.0 - 2015-06-22	25
3.81	1.3.1 - 2015-06-19	25
3.82	1.3.0 - 2015-06-19	25
3.83	1.2.0 - 2015-06-06	25
3.84	1.1.7 - 2015-03-19	26
3.85	1.1.6 - 2014-06-27	26
3.86	1.1.5 - 2013-11-10	26
3.87	1.1.4 - 2013-03-06	26
3.88	1.1.3 - 2012-12-17	26
3.89	1.1.2 - 2012-12-13	27
3.90	1.1.1 - 2012-12-05	27
3.91	1.1.0 - 2012-10-04	27
3.92	1.0.8 - 2010-04-07	27
3.93	1.0.5 - 2010-04-06	27
3.94	1.0.4 - 2010-04-06	28
3.95	1.0.0 - 2009-01-31	28
<b>4</b>	<b>tcod.bsp</b>	<b>29</b>
<b>5</b>	<b>tcod.console</b>	<b>33</b>
<b>6</b>	<b>tcod.event</b>	<b>43</b>
<b>7</b>	<b>tcod.image</b>	<b>51</b>
<b>8</b>	<b>tcod.map</b>	<b>55</b>
<b>9</b>	<b>tcod.noise</b>	<b>59</b>
<b>10</b>	<b>tcod.path</b>	<b>63</b>
<b>11</b>	<b>tcod.random</b>	<b>67</b>
<b>12</b>	<b>tcod.tileset</b>	<b>69</b>
<b>13</b>	<b>libtcodpy</b>	<b>71</b>
13.1	bsp	71
13.2	color	72
13.3	console	75
13.4	Event	84
13.5	sys	86
13.6	pathfinding	89
13.7	heightmap	91
13.8	image	97
13.9	line	98
13.10	map	100
13.11	mouse	101
13.12	namegen	101
13.13	noise	102
13.14	parser	103
13.15	random	103
13.16	struct	106
13.17	other	106
<b>14</b>	<b>tdl</b>	<b>109</b>
14.1	Getting Started	109

14.2	Indexing Consoles . . . . .	109
14.3	Drawing and Colors . . . . .	109
14.4	tdl API . . . . .	110
14.5	tdl.Console . . . . .	112
14.6	tdl.Window . . . . .	117
<b>15</b>	<b>tdl.event</b>	<b>121</b>
<b>16</b>	<b>tdl.map</b>	<b>127</b>
<b>17</b>	<b>tdl.noise</b>	<b>131</b>
<b>18</b>	<b>Indices and tables</b>	<b>133</b>
	<b>Python Module Index</b>	<b>135</b>
	<b>Index</b>	<b>137</b>

Contents:





Once installed, you'll be able to import the *tcod* and *libtcodpy* modules, as well as the deprecated *tdl* module.

Python 3.5 or above is required for a normal install. These instructions include installing Python if you don't have it yet.

There are known issues in very old versions of pip. If pip fails to install python-tcod then try updating it first.

## 1.1 Windows

First install a recent version of Python 3. Make sure Python is added to the Windows *PATH*.

If you don't already have it, then install the latest [Microsoft Visual C++ Redistributable](#). **vc\_redist.x86.exe** for a 32-bit install of Python, or **vc\_redist.x64.exe** for a 64-bit install. You'll need to keep this in mind when distributing any libtcod program to end-users.

Then to install python-tcod run the following from a Windows command line:

```
py -m pip install tcod
```

If Python was installed for all users then you may need to add the `--user` flag to pip.

## 1.2 MacOS

The latest version of python-tcod only supports MacOS 10.9 (Mavericks) or later.

First install a recent version of Python 3.

Then to install using pip in a user environment, use the following command:

```
python3 -m pip install --user tcod
```

## 1.3 Linux (Debian-based)

On Linux python-tcod will need to be built from source. You can run this command to download python-tcod's dependencies with apt:

```
sudo apt install build-essential python3-dev python3-pip python3-numpy libSDL2-dev_  
↳libffi-dev libomp5
```

If your GCC version is less than 6.1, or your SDL version is less than 2.0.5, then you will need to perform a distribution upgrade before continuing.

Once dependences are resolved you can build and install python-tcod using pip in a user environment:

```
python3 -m pip install --user tcod
```

## 1.4 Upgrading python-tcod

python-tcod is updated often, you can re-run pip with the `--upgrade` flag to ensure you have the latest version, for example:

```
python3 -m pip install --upgrade tcod
```

## 1.5 Upgrading from libtcodpy to python-tcod

*libtcodpy* is no longer maintained and using it can make it difficult to collaborate with developers across multiple operating systems, or to distribute to those platforms. New API features are only available on *python-tcod*.

You can recognise a libtcodpy program because it includes this file structure:

```
libtcodpy/  
libtcod.dll  
SDL2.dll
```

First make sure your libtcodpy project works in Python 3. libtcodpy already supports both 2 and 3 so you don't need to worry about updating it, but you will need to worry about bit-size. If you're using a 32-bit version of Python 2 then you'll need to upgrade to a 32-bit version of Python 3 until libtcodpy can be completely removed.

Once you've installed python-tcod you can safely delete the `libtcodpy/` folder and all DLL files of a libtcodpy program, python-tcod will seamlessly take the place of libtcodpy's API.

From then on anyone can follow the instructions to install python-tcod and your project will work for them regardless of their platform or bit-size.

## 1.6 Distributing

Once your project is finished, it can be distributed using [PyInstaller](#).

## 1.7 Python 2.7

While it's not recommended, you can still install *python-tcod* on *Python 2.7*.

Keep in mind the Python 2's end-of-life is the year 2020. You should not be starting any new projects in Python 2!

Follow the instructions for your platform normally. When it comes to install with pip, tell it to get python-tcod version 6:

```
python2 -m pip install tcod==6.0.7
```



**console defaults** The default values implied by any Console print or put functions which don't explicitly ask for them as parameters.

These have been deprecated since version 8.5.

**libcod-ffi** This is the *ffi* implementation of libcodpy, the original was made using *ctypes* which was more difficult to maintain.

*libcod-ffi* is now part of *python-tcod*.

**python-tcod** *python-tcod* is a superset of the *libcodpy* API. The major additions include class functionality in returned objects, no manual memory management, pickle-able objects, and *numpy* array attributes in most objects.

The *numpy* attributes in particular can be used to dramatically speed up the performance of your program compared to using *libcodpy*.

**python-tdl** *tdl* is a high-level wrapper over *libcodpy* although it now uses *python-tcod*, it doesn't do anything that you couldn't do yourself with just *libcodpy* and Python.

Currently no new features are planned for *tdl*, instead new features are added to *libcod* itself and then ported to *python-tcod*.

*python-tdl* and *libcodpy* are included in installations of *python-tcod*.

**libcodpy** *libcodpy* is more or less a direct port of *libcod*'s C API to Python. This caused a handful of issues including instances needing to be freed manually or a memory leak will occur, and some functions performing badly in Python due to the need to call them frequently.

These issues are fixed in *python-tcod* which implements the full *libcodpy* API. If *python-tcod* is installed then imports of *libcodpy* are aliased to the *tcod* module. So if you come across a project using the original *libcodpy* you can delete the *libcodpy/* folder and then *python-tcod* will load instead.



Changes relevant to the users of python-tcod are documented here.

This project adheres to [Semantic Versioning](#) since v2.0.0

### 3.1 Unreleased

### 3.2 11.3.0 - 2019-09-06

#### Added

- New attribute *Console.tiles2* is similar to *Console.tiles* but without an alpha channel.

### 3.3 11.2.2 - 2019-08-25

#### Fixed

- Fixed a regression preventing PyInstaller distributions from loading SDL2.

### 3.4 11.2.1 - 2019-08-25

### 3.5 11.2.0 - 2019-08-24

#### Added

- *tcod.path.dijkstra2d*: Computes Dijkstra from an arbitrary initial state.
- *tcod.path.hillclimb2d*: Returns a path from a distance array.

- *tcod.path.maxarray*: Creates arrays filled with maximum finite values.

**Fixed**

- Changing the tiles of an active tileset on OPENGL2 will no longer leave temporary artifact tiles.
- It's now harder to accidentally import tcod's internal modules.

### 3.6 11.1.2 - 2019-08-02

**Changed**

- Now bundles SDL 2.0.10 for Windows/MacOS.

**Fixed**

- Can now parse SDL 2.0.10 headers during installation without crashing.

### 3.7 11.1.1 - 2019-08-01

**Deprecated**

- Using an out-of-bounds index for field-of-view operations now raises a warning, which will later become an error.

**Fixed**

- Changing the tiles of an active tileset will now work correctly.

### 3.8 11.1.0 - 2019-07-05

**Added**

- You can now set the *TCOD\_RENDERER* and *TCOD\_VSYNC* environment variables to force specific options to be used. Example: `TCOD_RENDERER=sd12 TCOD_VSYNC=1`

**Changed**

- *tcod.sys\_set\_renderer* now raises an exception if it fails.

**Fixed**

- *tcod.console\_map\_ascii\_code\_to\_font* functions will now work when called before *tcod.console\_init\_root*.

### 3.9 11.0.2 - 2019-06-21

**Changed**

- You no longer need OpenGL to build python-tcod.



### 3.10 11.0.1 - 2019-06-21

#### Changed

- Better runtime checks for Windows dependencies should now give distinct errors depending on if the issue is SDL2 or missing redistributables.

#### Fixed

- Changed NumPy type hints from *np.array* to *np.ndarray* which should resolve issues.

### 3.11 11.0.0 - 2019-06-14

#### Changed

- *tcod.map.compute\_fov* now takes a 2-item tuple instead of separate *x* and *y* parameters. This causes less confusion over how axes are aligned.

### 3.12 10.1.1 - 2019-06-02

#### Changed

- Better string representations for *tcod.event.Event* subclasses.

#### Fixed

- Fixed regressions in text alignment for non-rectangle print functions.

### 3.13 10.1.0 - 2019-05-24

#### Added

- *tcod.console\_init\_root* now has an optional *vsync* parameter.

### 3.14 10.0.5 - 2019-05-17

#### Fixed

- Fixed shader compilation issues in the OPENGL2 renderer.
- Fallback fonts should fail less on Linux.

### 3.15 10.0.4 - 2019-05-17

#### Changed

- Now depends on cffi 0.12 or later.

#### Fixed

- *tcod.console\_init\_root* and *tcod.console\_set\_custom\_font* will raise exceptions instead of terminating.

- Fixed issues preventing *tcod.event* from working on 32-bit Windows.

### 3.16 10.0.3 - 2019-05-10

#### Fixed

- Corrected bounding box issues with the *Console.print\_box* method.

### 3.17 10.0.2 - 2019-04-26

#### Fixed

- Resolved Color warnings when importing *tcod*.
- When compiling, fixed a name conflict with endianness macros on FreeBSD.

### 3.18 10.0.1 - 2019-04-19

#### Fixed

- Fixed horizontal alignment for TrueType fonts.
- Fixed taking screenshots with the older SDL renderer.

### 3.19 10.0.0 - 2019-03-29

#### Added

- New *Console.tiles* array attribute.

#### Changed

- *Console.DTYPE* changed to add alpha to its color types.

#### Fixed

- Console printing was ignoring color codes at the beginning of a string.

### 3.20 9.3.0 - 2019-03-15

#### Added

- The SDL2/OPENGL2 renderers can potentially use a fall-back font when none are provided.
- New function *tcod.event.get\_mouse\_state*.
- New function *tcod.map.compute\_fov* lets you get a visibility array directly from a transparency array.

#### Deprecated

- The following functions and classes have been deprecated. - *tcod.Key* - *tcod.Mouse* - *tcod.mouse\_get\_status* - *tcod.console\_is\_window\_closed* - *tcod.console\_check\_for\_keypress* - *tcod.console\_wait\_for\_keypress* - *tcod.console\_delete* - *tcod.sys\_check\_for\_event* - *tcod.sys\_wait\_for\_event*
- The SDL, OPENGL, and GLSL renderers have been deprecated.
- Many libtcodpy functions have been marked with PendingDeprecationWarning's.

#### Fixed

- To be more compatible with libtcodpy *tcod.console\_init\_root* will default to the SDL render, but will raise warnings when an old renderer is used.

### 3.21 9.2.5 - 2019-03-04

#### Fixed

- Fixed *tcod.namegen\_generate\_custom*.

### 3.22 9.2.4 - 2019-03-02

#### Fixed

- The *tcod* package is has been marked as typed and will now work with MyPy.

### 3.23 9.2.3 - 2019-03-01

#### Deprecated

- The behavior for negative indexes on the new print functions may change in the future.
- Methods and functionality preventing *tcod.Color* from behaving like a tuple have been deprecated.

### 3.24 9.2.2 - 2019-02-26

#### Fixed

- *Console.print\_box* wasn't setting the background color by default.

### 3.25 9.2.1 - 2019-02-25

#### Fixed

- *tcod.sys\_get\_char\_size* fixed on the new renderers.

## 3.26 9.2.0 - 2019-02-24

### Added

- New `tcod.console.get_height_rect` function, which can be used to get the height of a print call without an existing console.
- New `tcod.tileset` module, with a `set_truetype_font` function.

### Fixed

- The new print methods now handle alignment according to how they were documented.
- `SDL2` and `OPENGL2` now support screenshots.
- Windows and MacOS builds now restrict exported `SDL2` symbols to only `SDL 2.0.5`; This will avoid hard to debug import errors when the wrong version of `SDL` is dynamically linked.
- The root console now starts with a white foreground.

## 3.27 9.1.0 - 2019-02-23

### Added

- Added the `tcod.random.MULTIPLY_WITH_CARRY` constant.

### Changed

- The overhead for warnings has been reduced when running Python with the optimize `-O` flag.
- `tcod.random.Random` now provides a default algorithm.

## 3.28 9.0.0 - 2019-02-17

### Changed

- New console methods now default to an `fg` and `bg` of `None` instead of white-on-black.

## 3.29 8.5.0 - 2019-02-15

### Added

- `tcod.console.Console` now supports `str` and `repr`.
- Added new Console methods which are independent from the console defaults.
- You can now give an array when initializing a `tcod.console.Console` instance.
- `Console.clear` can now take `ch`, `fg`, and `bg` parameters.

### Changed

- Updated `libtcod` to 1.10.6
- Printing generates more compact layouts.

### Deprecated

- Most `libtcodpy` console functions have been replaced by the `tcod.console` module.

- Deprecated the *set\_key\_color* functions. You can pass key colors to *Console.blit* instead.
- *Console.clear* should be given the colors to clear with as parameters, rather than by using *default\_fg* or *default\_bg*.
- Most functions which depend on console default values have been deprecated. The new deprecation warnings will give details on how to make default values explicit.

**Fixed**

- *tcod.console.Console.blit* was ignoring the key color set by *Console.set\_key\_color*.
- The *SDL2* and *OPENGL2* renders can now large numbers of tiles.

### 3.30 8.4.3 - 2019-02-06

**Changed**

- Updated libtcod to 1.10.5
- The *SDL2/OPENGL2* renderers will now auto-detect a custom fonts key-color.

### 3.31 8.4.2 - 2019-02-05

**Deprecated**

- The *tdl* module has been deprecated.
- The libtcodpy parser functions have been deprecated.

**Fixed**

- *tcod.image\_is\_pixel\_transparent* and *tcod.image\_get\_alpha* now return values.
- *Console.print\_frame* was clearing tiles outside if its bounds.
- The *FONT\_LAYOUT\_CP437* layout was incorrect.

### 3.32 8.4.1 - 2019-02-01

**Fixed**

- Window event types were not upper-case.
- Fixed regression where libtcodpy mouse wheel events unset mouse coordinates.

### 3.33 8.4.0 - 2019-01-31

**Added**

- Added *tcod.event* module, based off of the *sdlevent.py* shim.

**Changed**

- Updated libtcod to 1.10.3

**Fixed**

- Fixed libtcodpy *struct\_add\_value\_list* function.
- Use correct math for tile-based delta in mouse events.
- New renderers now support tile-based mouse coordinates.
- SDL2 renderer will now properly refresh after the window is resized.

### 3.34 8.3.2 - 2018-12-28

#### Fixed

- Fixed rare access violations for some functions which took strings as parameters, such as *tcod.console\_init\_root*.

### 3.35 8.3.1 - 2018-12-28

#### Fixed

- libtcodpy key and mouse functions will no longer accept the wrong types.
- The *new\_struct* method was not being called for libtcodpy's custom parsers.

### 3.36 8.3.0 - 2018-12-08

#### Added

- Added BSP traversal methods in *tcod.bsp* for parity with libtcodpy.

#### Deprecated

- Already deprecated bsp functions are now even more deprecated.

### 3.37 8.2.0 - 2018-11-27

#### Added

- New layout *tcod.FONT\_LAYOUT\_CP437*.

#### Changed

- Updated libtcod to 1.10.2
- *tcod.console\_print\_frame* and *Console.print\_frame* now support Unicode strings.

#### Deprecated

- Deprecated using bytes strings for all printing functions.

#### Fixed

- Console objects are now initialized with spaces. This fixes some blit operations.
- Unicode code-points above U+FFFF will now work on all platforms.

### 3.38 8.1.1 - 2018-11-16

#### Fixed

- Printing a frame with an empty string no longer displays a title bar.

### 3.39 8.1.0 - 2018-11-15

#### Changed

- Heightmap functions now support 'F\_CONTIGUOUS' arrays.
- `tcod.heightmap_new` now has an *order* parameter.
- Updated SDL to 2.0.9

#### Deprecated

- Deprecated heightmap functions which sample noise grids, this can be done using the *Noise.sample\_ogrid* method.

### 3.40 8.0.0 - 2018-11-02

#### Changed

- The default renderer can now be anything if not set manually.
- Better error message for when a font file isn't found.

### 3.41 7.0.1 - 2018-10-27

#### Fixed

- Building from source was failing because *console\_2tris.glsl\** was missing from source distributions.

### 3.42 7.0.0 - 2018-10-25

#### Added

- New *RENDERER\_SDL2* and *RENDERER\_OPENGL2* renderers.

#### Changed

- Updated libtcod to 1.9.0
- Now requires SDL 2.0.5, which is not trivially installable on Ubuntu 16.04 LTS.

#### Removed

- Dropped support for Python versions before 3.5
- Dropped support for MacOS versions before 10.9 Mavericks.

### 3.43 6.0.7 - 2018-10-24

**Fixed**

- The root console no longer loses track of buffers and console defaults on a renderer change.

### 3.44 6.0.6 - 2018-10-01

**Fixed**

- Replaced missing wheels for older and 32-bit versions of MacOS.

### 3.45 6.0.5 - 2018-09-28

**Fixed**

- Resolved CDefError error during source installs.

### 3.46 6.0.4 - 2018-09-11

**Fixed**

- tcod.Key right-hand modifiers are now set independently at initialization, instead of mirroring the left-hand modifier value.

### 3.47 6.0.3 - 2018-09-05

**Fixed**

- tcod.Key and tcod.Mouse no longer ignore initiation parameters.

### 3.48 6.0.2 - 2018-08-28

**Fixed**

- Fixed color constants missing at build-time.

### 3.49 6.0.1 - 2018-08-24

**Fixed**

- Source distributions were missing C++ source files.



### 3.50 6.0.0 - 2018-08-23

#### Changed

- Project renamed to tcod on PyPI.

#### Deprecated

- Passing bytes strings to libtcodpy print functions is deprecated.

#### Fixed

- Fixed libtcodpy print functions not accepting bytes strings.
- libtcod constants are now generated at build-time fixing static analysis tools.

### 3.51 5.0.1 - 2018-07-08

#### Fixed

- `tdl.event` no longer crashes with `StopIteration` on Python 3.7

### 3.52 5.0.0 - 2018-07-05

#### Changed

- `tcod.path`: all classes now use *shape* instead of *width* and *height*.
- `tcod.path` now respects NumPy array shape, instead of assuming that arrays need to be transposed from C memory order. From now on *x* and *y* mean 1st and 2nd axis. This doesn't affect non-NumPy code.
- `tcod.path` now has full support of non-contiguous memory.

### 3.53 4.6.1 - 2018-06-30

#### Added

- New function `tcod.line_where` for indexing NumPy arrays using a Bresenham line.

#### Deprecated

- Python 2.7 support will be dropped in the near future.

### 3.54 4.5.2 - 2018-06-29

#### Added

- New wheels for Python3.7 on Windows.

#### Fixed

- Arrays from `tcod.heightmap_new` are now properly zeroed out.

### 3.55 4.5.1 - 2018-06-23

#### Deprecated

- Deprecated all libtcodpy map functions.

#### Fixed

- `tcod.map_copy` could break the `tcod.map.Map` class.
- `tcod.map_clear` `transparent` and `walkable` parameters were reversed.
- When multiple SDL2 headers were installed, the wrong ones would be used when the library is built.
- Fails to build via pip unless Numpy is installed first.

### 3.56 4.5.0 - 2018-06-12

#### Changed

- Updated libtcod to v1.7.0
- Updated SDL to v2.0.8
- Error messages when failing to create an SDL window should be a less vague.
- You no longer need to initialize libtcod before you can print to an off-screen console.

#### Fixed

- Avoid crashes if the root console has a character code higher than expected.

#### Removed

- No more debug output when loading fonts.

### 3.57 4.4.0 - 2018-05-02

#### Added

- Added the libtcodpy module as an alias for `tcod`. Actual use of it is deprecated, it exists primarily for backward compatibility.
- Adding missing libtcodpy functions `console_has_mouse_focus` and `console_is_active`.

#### Changed

- Updated libtcod to v1.6.6

### 3.58 4.3.2 - 2018-03-18

#### Deprecated

- Deprecated the use of falsy console parameters with libtcodpy functions.

#### Fixed

- Fixed libtcodpy image functions not supporting falsy console parameters.

- Fixed tdl *Window.get\_char* method. (Kaczor2704)

### 3.59 4.3.1 - 2018-03-07

#### Fixed

- Fixed `ffi.api.FFIError` “unsupported expression: expected a simple numeric constant” error when building on platforms with an older `ffi` module and newer `SDL` headers.
- `tcod/tdl Map` and `Console` objects were not saving stride data when pickled.

### 3.60 4.3.0 - 2018-02-01

#### Added

- You can now set the `numpy` memory order on `tcod.console.Console`, `tcod.map.Map`, and `tdl.map.Map` objects well as from the `tcod.console_init_root` function.

#### Changed

- The `console_init_root title` parameter is now optional.

#### Fixed

- `OpenGL` renderer alpha blending is now consistent with all other render modes.

### 3.61 4.2.3 - 2018-01-06

#### Fixed

- Fixed `setup.py` regression that could prevent building outside of the git repository.

### 3.62 4.2.2 - 2018-01-06

#### Fixed

- The Windows dynamic linker will now prefer the bundled version of `SDL`. This fixes: “`ImportError: DLL load failed: The specified procedure could not be found.`”
- `key.c` is no longer set when `key.vk == KEY_TEXT`, this fixes a regression which was causing events to be heard twice in the `libtcod/Python` tutorial.

### 3.63 4.2.0 - 2018-01-02

#### Changed

- Updated `libtcod` backend to v1.6.4
- Updated `SDL` to v2.0.7 for Windows/MacOS.

#### Removed

- Source distributions no longer include tests, examples, or fonts. [Find these on GitHub.](#)

#### Fixed

- Fixed “final link failed: Nonrepresentable section on output” error when compiling for Linux.
- `tcod.console_init_root` defaults to the SDL renderer, other renderers cause issues with mouse movement events.

### 3.64 4.1.1 - 2017-11-02

#### Fixed

- Fixed `ConsoleBuffer.blit` regression.
- Console defaults corrected, the root console’s blend mode and alignment is the default value for newly made Console’s.
- You can give a byte string as a filename to load parsers.

### 3.65 4.1.0 - 2017-07-19

#### Added

- tdl Map class can now be pickled.

#### Changed

- Added protection to the `transparent`, `walkable`, and `fov` attributes in `tcod` and `tdl` Map classes, to prevent them from being accidentally overridden.
- `tcod` and `tdl` Map classes now use numpy arrays as their attributes.

### 3.66 4.0.1 - 2017-07-12

#### Fixed

- tdl: Fixed `NameError` in `set_fps`.

### 3.67 4.0.0 - 2017-07-08

#### Changed

- `tcod.bsp`: `BSP.split_recursive` parameter `random` is now `seed`.
- `tcod.console`: `Console.blit` parameters have been rearranged. Most of the parameters are now optional.
- `tcod.noise`: `Noise.__init__` parameter `rand` is now named `seed`.
- tdl: Changed `set_fps` paramter name to `fps`.

#### Fixed

- `tcod.bsp`: Corrected spelling of `max_vertical_ratio`.

### 3.68 3.2.0 - 2017-07-04

#### Changed

- Merged libtcod-ffi dependency with TDL.

#### Fixed

- Fixed boolean related crashes with Key 'text' events.
- tdl.noise: Fixed crash when given a negative seed. As well as cases where an instance could lose its seed being pickled.

### 3.69 3.1.0 - 2017-05-28

#### Added

- You can now pass tdl Console instances as parameters to libtcod-ffi functions expecting a tcod Console.

#### Changed

- Dependencies updated: *libtcod-ffi* >=2.5.0, <3
- The *Console.tcod\_console* attribute is being renamed to *Console.console\_c*.

#### Deprecated

- The tdl.noise and tdl.map modules will be deprecated in the future.

#### Fixed

- Resolved crash-on-exit issues for Windows platforms.

### 3.70 3.0.2 - 2017-04-13

#### Changed

- Dependencies updated: *libtcod-ffi* >=2.4.3, <3
- You can now create Console instances before a call to *tdl.init*.

#### Removed

- Dropped support for Python 3.3

#### Fixed

- Resolved issues with MacOS builds.
- 'OpenGL' and 'GLSL' renderers work again.

### 3.71 3.0.1 - 2017-03-22

#### Changed

- *KeyEvent*'s with *text* now have all their modifier keys set to False.

#### Fixed

- Undefined behaviour in text events caused crashes on 32-bit builds.

### 3.72 3.0.0 - 2017-03-21

#### Added

- *KeyEvent* supports libtcod text and meta keys.

#### Changed

- *KeyEvent* parameters have been moved.
- This version requires *libtcod-ffi*  $\geq 2.3.0$ .

#### Deprecated

- *KeyEvent* camel cased attribute names are deprecated.

#### Fixed

- Crashes with key-codes undefined by libtcod.
- *tdl.map* typedef issues with libtcod-ffi.

### 3.73 2.0.1 - 2017-02-22

#### Fixed

- *tdl.init* renderer was defaulted to OpenGL which is not supported in the current version of libtcod.

### 3.74 2.0.0 - 2017-02-15

#### Changed

- Dependencies updated, *tdl* now requires libtcod-ffi 2.x.x
- Some event behaviours have changed with SDL2, event keys might be different than what you expect.

#### Removed

- Key repeat functions were removed from SDL2. *set\_key\_repeat* is now stubbed, and does nothing.

### 3.75 1.6.0 - 2016-11-18

- *Console.blit* methods can now take *fg\_alpha* and *bg\_alpha* parameters.

### 3.76 1.5.3 - 2016-06-04

- *set\_font* no longer crashes when loading a file without the implied font size in its name

### 3.77 1.5.2 - 2016-03-11

- Fixed non-square Map instances

### 3.78 1.5.1 - 2015-12-20

- Fixed errors with Unicode and non-Unicode literals on Python 2
- Fixed attribute error in compute\_fov

### 3.79 1.5.0 - 2015-07-13

- python-tdl distributions are now universal builds
- New Map class
- map.bresenham now returns a list
- This release will require libtcod-ffi v0.2.3 or later

### 3.80 1.4.0 - 2015-06-22

- The DLL's have been moved into another library which you can find at <https://github.com/HexDecimal/libtcod-ffi> You can use this library to have some raw access to libtcod if you want. Plus it can be used alongside TDL.
- The libtcod console objects in Console instances have been made public.
- Added tdl.event.wait function. This function can called with a timeout and can automatically call tdl.flush.

### 3.81 1.3.1 - 2015-06-19

- Fixed pathfinding regressions.

### 3.82 1.3.0 - 2015-06-19

- Updated backend to use python-ffi instead of ctypes. This gives decent boost to speed in CPython and a drastic to boost in speed in PyPy.

### 3.83 1.2.0 - 2015-06-06

- The set\_colors method now changes the default colors used by the draw\_\* methods. You can use Python's Ellipsis to explicitly select default colors this way.
- Functions and Methods renamed to match Python's style-guide PEP 8, the old function names still exist and are deprecated.

- The fgcolor and bgcolor parameters have been shortened to fg and bg.

### 3.84 1.1.7 - 2015-03-19

- Noise generator now seeds properly.
- The OS event queue will now be handled during a call to `tdl.flush`. This prevents a common newbie programmer hang where events are handled infrequently during long animations, simulations, or early development.
- Fixed a major bug that would cause a crash in later versions of Python 3

### 3.85 1.1.6 - 2014-06-27

- Fixed a race condition when importing on some platforms.
- Fixed a type issue with `quickFOV` on Linux.
- Added a `bresenham` function to the `tdl.map` module.

### 3.86 1.1.5 - 2013-11-10

- A for loop can iterate over all coordinates of a Console.
- `drawStr` can be configured to scroll or raise an error.
- You can now configure or disable key repeating with `tdl.event.setKeyRepeat`
- Typewriter class removed, use a `Window` instance for the same functionality.
- `setColors` method fixed.

### 3.87 1.1.4 - 2013-03-06

- Merged the `Typewriter` and `MetaConsole` classes, You now have a virtual cursor with `Console` and `Window` objects.
- Fixed the `clear` method on the `Window` class.
- Fixed screenshot function.
- Fixed some drawing operations with unchanging backgrounds.
- Instances of `Console` and `Noise` can be pickled and copied.
- Added `KeyEvent.keychar`
- Fixed `event.keyWait`, and now converts window closed events into `Alt+F4`.

### 3.88 1.1.3 - 2012-12-17

- Some of the `setFont` parameters were incorrectly labeled and documented.
- `setFont` can auto-detect tilesets if the font sizes are in the filenames.



- Added some X11 unicode tilesets, including unifont.

### **3.89 1.1.2 - 2012-12-13**

- Window title now defaults to the running scripts filename.
- Fixed incorrect deltaTime for App.update
- App will no longer call tdl.flush on its own, you'll need to call this yourself.
- tdl.noise module added.
- clear method now defaults to black on black.

### **3.90 1.1.1 - 2012-12-05**

- Map submodule added with AStar class and quickFOV function.
- New Typewriter class.
- Most console functions can use Python-style negative indexes now.
- New App.runOnce method.
- Rectangle geometry is less strict.

### **3.91 1.1.0 - 2012-10-04**

- KeyEvent.keyname is now KeyEvent.key
- MouseButtonEvent.button now behaves like KeyEvent.keyname does.
- event.App class added.
- Drawing methods no longer have a default for the character parameter.
- KeyEvent.ctrl is now KeyEvent.control

### **3.92 1.0.8 - 2010-04-07**

- No longer works in Python 2.5 but now works in 3.x and has been partly tested.
- Many bug fixes.

### **3.93 1.0.5 - 2010-04-06**

- Got rid of setuptools dependency, this will make it much more compatible with Python 3.x
- Fixed a typo with the MacOS library import.

### 3.94 1.0.4 - 2010-04-06

- All constant colors (C\_\*) have been removed, they may be put back in later.
- Made some type assertion failures show the value they received to help in general debugging. Still working on it.
- Added MacOS and 64-bit Linux support.

### 3.95 1.0.0 - 2009-01-31

- First public release.

The following example shows how to traverse the BSP tree using Python. This assumes *create\_room* and *connect\_rooms* will be replaced by custom code.

Example:

```
import tcod.bsp

bsp = tcod.bsp.BSP(x=0, y=0, width=80, height=60)
bsp.split_recursive(
    depth=5,
    min_width=3,
    min_height=3,
    max_horizontal_ratio=1.5,
    max_vertical_ratio=1.5,
)

# In pre order, leaf nodes are visited before the nodes that connect them.
for node in bsp.pre_order():
    if node.children:
        node1, node2 = node.children
        print('Connect the rooms:\n%s\n%s' % (node1, node2))
    else:
        print('Dig a room for %s.' % node)
```

**class** `tcod.bsp.BSP` (*x: int, y: int, width: int, height: int*)

A binary space partitioning tree which can be used for simple dungeon generation.

**x**

Rectangle left coordinate.

**Type** `int`

**y**

Rectangle top coordinate.

**Type** `int`

**width**

Rectangle width.

**Type** `int`

**height**

Rectangle height.

**Type** `int`

**level**

This nodes depth.

**Type** `int`

**position**

The integer of where the node was split.

**Type** `int`

**horizontal**

This nodes split orientation.

**Type** `bool`

**parent**

This nodes parent or None

**Type** `Optional[BSP]`

**children**

A tuple of (left, right) BSP instances, or an empty tuple if this BSP has no children.

**Type** `Union[Tuple[()], Tuple[BSP, BSP]]`

**Parameters**

- **x** (`int`) – Rectangle left coordinate.
- **y** (`int`) – Rectangle top coordinate.
- **width** (`int`) – Rectangle width.
- **height** (`int`) – Rectangle height.

`__str__` () → `str`

Provide a useful readout when printed.

`contains` (`x: int, y: int`) → `bool`

Returns True if this node contains these coordinates.

**Parameters**

- **x** (`int`) – X position to check.
- **y** (`int`) – Y position to check.

**Returns**

**True if this node contains these coordinates.** Otherwise False.

**Return type** `bool`

`find_node` (`x: int, y: int`) → `Optional[tcod.bsp.BSP]`

Return the deepest node which contains these coordinates.

**Returns** BSP object or None.

**Return type** Optional[*BSP*]

**in\_order** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in order.

New in version 8.3.

**inverted\_level\_order** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in inverse level order.

New in version 8.3.

**level\_order** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in level order.

New in version 8.3.

**post\_order** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in post order.

New in version 8.3.

**pre\_order** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in pre order.

New in version 8.3.

**split\_once** (*horizontal: bool, position: int*) → None

Split this partition into 2 sub-partitions.

#### Parameters

- **horizontal** (*bool*) –
- **position** (*int*) –

**split\_recursive** (*depth: int, min\_width: int, min\_height: int, max\_horizontal\_ratio: float, max\_vertical\_ratio: float, seed: Optional[tcod.random.Random] = None*) → None

Divide this partition recursively.

#### Parameters

- **depth** (*int*) – The maximum depth to divide this object recursively.
- **min\_width** (*int*) – The minimum width of any individual partition.
- **min\_height** (*int*) – The minimum height of any individual partition.
- **max\_horizontal\_ratio** (*float*) – Prevent creating a horizontal ratio more extreme than this.
- **max\_vertical\_ratio** (*float*) – Prevent creating a vertical ratio more extreme than this.
- **seed** (*Optional[tcod.random.Random]*) – The random number generator to use.

**walk** () → Iterator[tcod.bsp.BSP]

Iterate over this BSP's hierarchy in pre order.

Deprecated since version 2.3: Use *pre\_order* instead.



libtcod works with a special ‘root’ console. You create this console using the `tcod.console_init_root` function. Usually after setting the font with `console_set_custom_font` first.

Example:

```
# Make sure 'arial10x10.png' is in the same directory as this script.
import tcod
import tcod.event

# Setup the font.
tcod.console_set_custom_font(
    "arial10x10.png",
    tcod.FONT_LAYOUT_TCOD | tcod.FONT_TYPE_GREYSCALE,
)
# Initialize the root console in a context.
with tcod.console_init_root(80, 60, order="F") as root_console:
    root_console.print_(x=0, y=0, string='Hello World!')
    while True:
        tcod.console_flush() # Show the console.
        for event in tcod.event.wait():
            if event.type == "QUIT":
                raise SystemExit()
# The libtcod window will be closed at the end of this with-block.
```

**class** `tcod.console.Console`(*width*: int, *height*: int, *order*: str = 'C', *buffer*: Optional[`numpy.ndarray`] = None)

A console object containing a grid of characters with foreground/background colors.

*width* and *height* are the size of the console (in tiles.)

*order* determines how the axes of NumPy array attributes are arranged. *order*="F" will swap the first two axes which allows for more intuitive [*x*, *y*] indexing.

With *buffer* the console can be initialized from another array. The *buffer* should be compatible with the *width*, *height*, and *order* given; and should also have a dtype compatible with `Console.DTYPE`.

Changed in version 4.3: Added *order* parameter.

Changed in version 8.5: Added *buffer*, *copy*, and default parameters. Arrays are initialized as if the *clear* method was called.

Changed in version 10.0: *DTYPE* changed, *buffer* now requires colors with an alpha channel.

### **console\_c**

A python-cffi “TCOD\_Console\*” object.

### **DTYPE**

A class attribute which provides a dtype compatible with this class.

```
[("ch", np.intc), ("fg", "(4,)u1"), ("bg", "(4,)u1")]
```

Example:

```
>>> buffer = np.zeros(
...     shape=(20, 3),
...     dtype=tcod.console.Console.DTYPE,
...     order="F",
... )
>>> buffer["ch"] = ord(' ')
>>> buffer["ch"][:, 1] = ord('x')
>>> c = tcod.console.Console(20, 3, order="F", buffer=buffer)
>>> print(c)
<          |
|xxxxxxxxxxxxxxxxxxxxxxxx|
|          >
```

New in version 8.5.

Changed in version 10.0: Added an alpha channel to the color types.

**\_\_bool\_\_**() → bool

Returns False if this is the root console.

This mimics libtcodpy behavior.

**\_\_enter\_\_**() → tcod.console.Console

Returns this console in a managed context.

When the root console is used as a context, the graphical window will close once the context is left as if *tcod.console\_delete* was called on it.

This is useful for some Python IDE’s like IDLE, where the window would not be closed on its own otherwise.

### **See also:**

*tcod.console\_init\_root*

**\_\_exit\_\_**(\*args) → None

Closes the graphical window on exit.

Some tcod functions may have undefined behavior after this point.

**\_\_repr\_\_**() → str

Return a string representation of this console.

**\_\_str\_\_**() → str

Return a simplified representation of this consoles contents.



**blit** (*dest: tcod.console.Console, dest\_x: int = 0, dest\_y: int = 0, src\_x: int = 0, src\_y: int = 0, width: int = 0, height: int = 0, fg\_alpha: float = 1.0, bg\_alpha: float = 1.0, key\_color: Optional[Tuple[int, int, int]] = None*) → None  
 Blit from this console onto the *dest* console.

#### Parameters

- **dest** (*Console*) – The destination console to blit onto.
- **dest\_x** (*int*) – Leftmost coordinate of the destination console.
- **dest\_y** (*int*) – Topmost coordinate of the destination console.
- **src\_x** (*int*) – X coordinate from this console to blit, from the left.
- **src\_y** (*int*) – Y coordinate from this console to blit, from the top.
- **width** (*int*) – The width of the region to blit.  
 If this is 0 the maximum possible width will be used.
- **height** (*int*) – The height of the region to blit.  
 If this is 0 the maximum possible height will be used.
- **fg\_alpha** (*float*) – Foreground color alpha vaule.
- **bg\_alpha** (*float*) – Background color alpha vaule.
- **key\_color** (*Optional[Tuple[int, int, int]]*) – None, or a (red, green, blue) tuple with values of 0-255.

Changed in version 4.0: Parameters were rearranged and made optional.

Previously they were: (*x, y, width, height, dest, dest\_x, dest\_y, \**)

**clear** (*ch: int = 32, fg: Tuple[int, int, int] = Ellipsis, bg: Tuple[int, int, int] = Ellipsis*) → None  
 Reset all values in this console to a single value.

*ch* is the character to clear the console with. Defaults to the space character.

*fg* and *bg* are the colors to clear the console with. Defaults to white-on-black if the console defaults are untouched.

---

**Note:** If *fg/bg* are not set, they will default to *default\_fg/default\_bg*. However, default values other than white-on-back are deprecated.

---

Changed in version 8.5: Added the *ch*, *fg*, and *bg* parameters. Non-white-on-black default values are deprecated.

**draw\_frame** (*x: int, y: int, width: int, height: int, title: str = "", clear: bool = True, fg: Optional[Tuple[int, int, int]] = None, bg: Optional[Tuple[int, int, int]] = None, bg\_blend: int = 1*) → None  
 Draw a framed rectangle with an optional title.

*x* and *y* are the starting tile, with 0, 0 as the upper-left corner of the console. You can use negative numbers if you want to start printing relative to the bottom-right corner, but this behavior may change in future versions.

*width* and *height* determine the size of the frame.

*title* is a Unicode string.

If *clear* is True than the region inside of the frame will be cleared.

*fg* and *bg* are the foreground text color and background tile color respectfully. This is a 3-item tuple with (r, g, b) color values from 0 to 255. These parameters can also be set to *None* to leave the colors unchanged.

*bg\_blend* is the blend type used by libtcod.

New in version 8.5.

Changed in version 9.0: *fg* and *bg* now default to *None* instead of white-on-black.

**draw\_rect** (*x*: int, *y*: int, *width*: int, *height*: int, *ch*: int, *fg*: Optional[Tuple[int, int, int]] = None, *bg*: Optional[Tuple[int, int, int]] = None, *bg\_blend*: int = 1) → None  
Draw characters and colors over a rectangular region.

*x* and *y* are the starting tile, with 0, 0 as the upper-left corner of the console. You can use negative numbers if you want to start printing relative to the bottom-right corner, but this behavior may change in future versions.

*width* and *height* determine the size of the rectangle.

*ch* is a Unicode integer. You can use 0 to leave the current characters unchanged.

*fg* and *bg* are the foreground text color and background tile color respectfully. This is a 3-item tuple with (r, g, b) color values from 0 to 255. These parameters can also be set to *None* to leave the colors unchanged.

*bg\_blend* is the blend type used by libtcod.

New in version 8.5.

Changed in version 9.0: *fg* and *bg* now default to *None* instead of white-on-black.

**get\_height\_rect** (*x*: int, *y*: int, *width*: int, *height*: int, *string*: str) → int  
Return the height of this text word-wrapped into this rectangle.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **width** (*int*) – Maximum width to render the text.
- **height** (*int*) – Maximum lines to render the text.
- **string** (*str*) – A Unicode string.

**Returns** The number of lines of text once word-wrapped.

**Return type** int

**hline** (*x*: int, *y*: int, *width*: int, *bg\_blend*: int = 13) → None  
Draw a horizontal line on the console.

This always uses ord(‘-’), the horizontal line character.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **width** (*int*) – The horizontal length of this line.
- **bg\_blend** (*int*) – The background blending flag.

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.draw_rect` instead, calling this function will print a warning detailing which default values need to be made explicit.

**print** (*x*: int, *y*: int, *string*: str, *fg*: Optional[Tuple[int, int, int]] = None, *bg*: Optional[Tuple[int, int, int]] = None, *bg\_blend*: int = 1, *alignment*: int = 0) → None  
 Print a string on a console with manual line breaks.

*x* and *y* are the starting tile, with 0, 0 as the upper-left corner of the console. You can use negative numbers if you want to start printing relative to the bottom-right corner, but this behavior may change in future versions.

*string* is a Unicode string which may include color control characters. Strings which are too long will be truncated until the next newline character "\n".

*fg* and *bg* are the foreground text color and background tile color respectfully. This is a 3-item tuple with (r, g, b) color values from 0 to 255. These parameters can also be set to *None* to leave the colors unchanged.

*bg\_blend* is the blend type used by libtcod.

*alignment* can be *tcod.LEFT*, *tcod.CENTER*, or *tcod.RIGHT*.

New in version 8.5.

Changed in version 9.0: *fg* and *bg* now default to *None* instead of white-on-black.

**print\_** (*x*: int, *y*: int, *string*: str, *bg\_blend*: int = 13, *alignment*: Optional[int] = None) → None  
 Print a color formatted string on a console.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **string** (*str*) – A Unicode string optionally using color codes.
- **bg\_blend** (*int*) – Blending mode to use, defaults to BKGND\_DEFAULT.
- **alignment** (*Optional[int]*) – Text alignment.

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.print` instead, calling this function will print a warning detailing which default values need to be made explicit.

**print\_box** (*x*: int, *y*: int, *width*: int, *height*: int, *string*: str, *fg*: Optional[Tuple[int, int, int]] = None, *bg*: Optional[Tuple[int, int, int]] = None, *bg\_blend*: int = 1, *alignment*: int = 0) → int  
 Print a string constrained to a rectangle and return the height.

*x* and *y* are the starting tile, with 0, 0 as the upper-left corner of the console. You can use negative numbers if you want to start printing relative to the bottom-right corner, but this behavior may change in future versions.

*width* and *height* determine the bounds of the rectangle, the text will automatically be word-wrapped to fit within these bounds.

*string* is a Unicode string which may include color control characters.

*fg* and *bg* are the foreground text color and background tile color respectfully. This is a 3-item tuple with (r, g, b) color values from 0 to 255. These parameters can also be set to *None* to leave the colors unchanged.

*bg\_blend* is the blend type used by libtcod.

*alignment* can be *tcod.LEFT*, *tcod.CENTER*, or *tcod.RIGHT*.

Returns the actual height of the printed area.

New in version 8.5.

Changed in version 9.0: *fg* and *bg* now default to *None* instead of white-on-black.

**print\_frame** (*x: int, y: int, width: int, height: int, string: str = "", clear: bool = True, bg\_blend: int = 13*) → None

Draw a framed rectangle with optional text.

This uses the default background color and blend mode to fill the rectangle and the default foreground to draw the outline.

*string* will be printed on the inside of the rectangle, word-wrapped. If *string* is empty then no title will be drawn.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **width** (*int*) – The width of the frame.
- **height** (*int*) – The height of the frame.
- **string** (*str*) – A Unicode string to print.
- **clear** (*bool*) – If True all text in the affected area will be removed.
- **bg\_blend** (*int*) – The background blending flag.

Changed in version 8.2: Now supports Unicode strings.

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.draw_frame` instead, calling this function will print a warning detailing which default values need to be made explicit.

**print\_rect** (*x: int, y: int, width: int, height: int, string: str, bg\_blend: int = 13, alignment: Optional[int] = None*) → int

Print a string constrained to a rectangle.

If  $h > 0$  and the bottom of the rectangle is reached, the string is truncated. If  $h = 0$ , the string is only truncated if it reaches the bottom of the console.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **width** (*int*) – Maximum width to render the text.
- **height** (*int*) – Maximum lines to render the text.
- **string** (*str*) – A Unicode string.
- **bg\_blend** (*int*) – Background blending flag.
- **alignment** (*Optional[int]*) – Alignment flag.

**Returns** The number of lines of text once word-wrapped.

#### Return type int

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.print_box` instead, calling this function will print a warning detailing which default values need to be made explicit.

**put\_char** (*x: int, y: int, ch: int, bg\_blend: int = 13*) → None

Draw the character *c* at *x,y* using the default colors and a blend mode.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **ch** (*int*) – Character code to draw. Must be in integer form.
- **bg\_blend** (*int*) – Blending mode to use, defaults to BKGND\_DEFAULT.

**rect** (*x: int, y: int, width: int, height: int, clear: bool, bg\_blend: int = 13*) → None  
 Draw a the background color on a rect optionally clearing the text.

If *clear* is True the affected tiles are changed to space character.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **width** (*int*) – Maximum width to render the text.
- **height** (*int*) – Maximum lines to render the text.
- **clear** (*bool*) – If True all text in the affected area will be removed.
- **bg\_blend** (*int*) – Background blending flag.

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.draw_rect` instead, calling this function will print a warning detailing which default values need to be made explicit.

**set\_key\_color** (*color: Optional[Tuple[int, int, int]]*) → None  
 Set a consoles blit transparent color.

*color* is the (r, g, b) color, or None to disable key color.

Deprecated since version 8.5: Pass the key color to `Console.blit` instead of calling this function.

**vline** (*x: int, y: int, height: int, bg\_blend: int = 13*) → None  
 Draw a vertical line on the console.

This always uses ord('|'), the vertical line character.

#### Parameters

- **x** (*int*) – The x coordinate from the left.
- **y** (*int*) – The y coordinate from the top.
- **height** (*int*) – The horizontal length of this line.
- **bg\_blend** (*int*) – The background blending flag.

Deprecated since version 8.5: Console methods which depend on console defaults have been deprecated. Use `Console.draw_rect` instead, calling this function will print a warning detailing which default values need to be made explicit.

#### bg

A uint8 array with the shape (height, width, 3).

You can change the consoles background colors by using this array.

Index this array with `console.bg[i, j, channel]` # `order='C'` or `console.bg[x, y, channel]` # `order='F'`.

#### ch

An integer array with the shape (height, width).

You can change the consoles character codes by using this array.

Index this array with `console.ch[i, j]` # order='C' or `console.ch[x, y]` # order='F'.

**default\_alignment**

The default text alignment.

**Type** `int`

**default\_bg**

The default background color.

**Type** `Tuple[int, int, int]`

**default\_bg\_blend**

The default blending mode.

**Type** `int`

**default\_fg**

The default foreground color.

**Type** `Tuple[int, int, int]`

**fg**

A uint8 array with the shape (height, width, 3).

You can change the consoles foreground colors by using this array.

Index this array with `console.fg[i, j, channel]` # order='C' or `console.fg[x, y, channel]` # order='F'.

**height**

The height of this Console. (read-only)

**Type** `int`

**tiles**

An array of this consoles raw tile data.

This acts as a combination of the *ch*, *fg*, and *bg* attributes. Colors include an alpha channel but how alpha works is currently undefined.

**Example**

```
>>> con = tcod.console.Console(10, 2, order="F")
>>> con.tiles[0, 0] = (
...     ord("X"),
...     (*tcod.white, 255),
...     (*tcod.black, 255),
... )
>>> con.tiles[0, 0]
(88, [255, 255, 255, 255], [ 0, 0, 0, 255])
```

New in version 10.0.

**tiles2**

An array of this consoles tile data without the alpha channel.

## Example

```
>>> con = tcod.console.Console(10, 2, order="F")
>>> con.tiles2[0, 0] = ord("@"), tcod.yellow, tcod.black
>>> con.tiles2[0, 0]
(64, [255, 255, 0], [0, 0, 0])
>>> con.tiles2["bg"] = tcod.blue
>>> con.tiles2[0, 0]
(64, [255, 255, 0], [ 0, 0, 255])
```

New in version 11.3.

### **width**

The width of this Console. (read-only)

**Type** `int`

`tcod.console.get_height_rect` (*width: int, string: str*) → `int`

Return the number of lines which would be printed from these parameters.

*width* is the width of the print boundary.

*string* is a Unicode string which may include color control characters.

New in version 9.2.





A light-weight implementation of event handling built on calls to SDL.

Many event constants are derived directly from SDL. For example: `tcod.event.K_UP` and `tcod.event.SCANCODE_A` refer to SDL's `SDLK_UP` and `SDL_SCANCODE_A` respectfully. [See this table for all of SDL's keyboard constants.](#)

Printing any event will tell you its attributes in a human readable format. An events type attribute if omitted is just the classes name with all letters upper-case. Do not use `isinstance` to tell events apart as that method won't be forward compatible.

As a general guideline, you should use `KeyboardEvent.sym` for command inputs, and `TextInput.text` for name entry fields.

Remember to add the line `import tcod.event`, as importing this module is not implied by `import tcod`.

New in version 8.4.

```
class tcod.event.Point (x, y)
```

**x**  
Alias for field number 0

**y**  
Alias for field number 1

```
class tcod.event.Event (type: Optional[str] = None)
  The base event class.
```

**type**  
This events type.

**Type** str

**sdl\_event**  
When available, this holds a python-cffi 'SDL\_Event\*' pointer. All sub-classes have this attribute.

**classmethod from\_sdl\_event** (sdl\_event: Any) → Any  
Return a class instance from a python-cffi 'SDL\_Event\*' pointer.

**class** `tcod.event.Quit` (*type: Optional[str] = None*)

An application quit request event.

For more info on when this event is triggered see: [https://wiki.libsdl.org/SDL\\_EventType#SDL\\_QUIT](https://wiki.libsdl.org/SDL_EventType#SDL_QUIT)

**type**

Always “QUIT”.

**Type** `str`

**classmethod** `from_sdl_event` (*sdl\_event: Any*) → `tcod.event.Quit`

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

**class** `tcod.event.KeyboardEvent` (*scancode: int, sym: int, mod: int, repeat: bool = False*)

**type**

Will be “KEYDOWN” or “KEYUP”, depending on the event.

**Type** `str`

**scancode**

The keyboard scan-code, this is the physical location of the key on the keyboard rather than the keys symbol.

**Type** `int`

**sym**

The keyboard symbol.

**Type** `int`

**mod**

A bitmask of the currently held modifier keys.

You can use the following to check if a modifier key is held:

- `tcod.event.KMOD_LSHIFT` Left shift bit.
- `tcod.event.KMOD_RSHIFT` Right shift bit.
- `tcod.event.KMOD_LCTRL` Left control bit.
- `tcod.event.KMOD_RCTRL` Right control bit.
- `tcod.event.KMOD_LALT` Left alt bit.
- `tcod.event.KMOD_RALT` Right alt bit.
- `tcod.event.KMOD_LGUI` Left meta key bit.
- `tcod.event.KMOD_RGUI` Right meta key bit.
- `tcod.event.KMOD_SHIFT` `tcod.event.KMOD_LSHIFT | tcod.event.KMOD_RSHIFT`
- `tcod.event.KMOD_CTRL` `tcod.event.KMOD_LCTRL | tcod.event.KMOD_RCTRL`
- `tcod.event.KMOD_ALT` `tcod.event.KMOD_LALT | tcod.event.KMOD_RALT`
- `tcod.event.KMOD_GUI` `tcod.event.KMOD_LGUI | tcod.event.KMOD_RGUI`
- `tcod.event.KMOD_NUM` Num lock bit.
- `tcod.event.KMOD_CAPS` Caps lock bit.
- `tcod.event.KMOD_MODE` AltGr key bit.

For example, if shift is held then `event.mod & tcod.event.KMOD_SHIFT` will evaluate to a true value.

**Type** `int`

**repeat**

True if this event exists because of key repeat.

**Type** `bool`

**classmethod** `from_sdl_event` (*sdl\_event: Any*) → Any

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

**class** `tcod.event.KeyDown` (*scancode: int, sym: int, mod: int, repeat: bool = False*)

**class** `tcod.event.KeyUp` (*scancode: int, sym: int, mod: int, repeat: bool = False*)

**class** `tcod.event.MouseMotion` (*pixel: Tuple[int, int] = (0, 0), pixel\_motion: Tuple[int, int] = (0, 0), tile: Tuple[int, int] = (0, 0), tile\_motion: Tuple[int, int] = (0, 0), state: int = 0*)

**type**

Always “MOUSEMOTION”.

**Type** `str`

**pixel**

The pixel coordinates of the mouse.

**Type** `Point`

**pixel\_motion**

The pixel delta.

**Type** `Point`

**tile**

The integer tile coordinates of the mouse on the screen.

**Type** `Point`

**tile\_motion**

The integer tile delta.

**Type** `Point`

**state**

A bitmask of which mouse buttons are currently held.

Will be a combination of the following names:

- `tcod.event.BUTTON_LMASK`
- `tcod.event.BUTTON_MMASK`
- `tcod.event.BUTTON_RMASK`
- `tcod.event.BUTTON_X1MASK`
- `tcod.event.BUTTON_X2MASK`

**Type** `int`

**classmethod** `from_sdl_event` (*sdl\_event: Any*) → `tcod.event.MouseMotion`

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

```
class tcod.event.MouseButtonEvent (pixel: Tuple[int, int] = (0, 0), tile: Tuple[int, int] = (0, 0),  
                                     button: int = 0)
```

**type**  
Will be “MOUSEBUTTONDOWN” or “MOUSEBUTTONUP”, depending on the event.

**Type** *str*

**pixel**  
The pixel coordinates of the mouse.

**Type** *Point*

**tile**  
The integer tile coordinates of the mouse on the screen.

**Type** *Point*

**button**  
Which mouse button.

This will be one of the following names:

- `tcod.event.BUTTON_LEFT`
- `tcod.event.BUTTON_MIDDLE`
- `tcod.event.BUTTON_RIGHT`
- `tcod.event.BUTTON_X1`
- `tcod.event.BUTTON_X2`

**Type** *int*

**classmethod** `from_sdl_event` (*sdl\_event: Any*) → *Any*  
Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

```
class tcod.event.MouseDown (pixel: Tuple[int, int] = (0, 0), tile: Tuple[int, int] = (0, 0),  
                             button: int = 0)  
Same as MouseButtonEvent but with type="MouseDown".
```

```
class tcod.event.MouseButtonUp (pixel: Tuple[int, int] = (0, 0), tile: Tuple[int, int] = (0, 0), button:  
                                  int = 0)  
Same as MouseButtonEvent but with type="MouseButtonUp".
```

```
class tcod.event.MouseWheel (x: int, y: int, flipped: bool = False)
```

**type**  
Always “MOUSEWHEEL”.

**Type** *str*

**x**  
Horizontal scrolling. A positive value means scrolling right.

**Type** *int*

**y**  
Vertical scrolling. A positive value means scrolling away from the user.

**Type** *int*

**flipped**

If True then the values of *x* and *y* are the opposite of their usual values. This depends on the settings of the Operating System.

**Type** bool

**classmethod from\_sdl\_event** (*sdl\_event: Any*) → tcod.event.MouseWheel

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

**class** tcod.event.**TextInput** (*text: str*)

**type**

Always “TEXTINPUT”.

**Type** str

**text**

A Unicode string with the input.

**Type** str

**classmethod from\_sdl\_event** (*sdl\_event: Any*) → tcod.event.TextInput

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

**class** tcod.event.**WindowEvent** (*type: Optional[str] = None*)

**type**

A window event could mean various event types.

**Type** str

**classmethod from\_sdl\_event** (*sdl\_event: Any*) → Any

Return a class instance from a python-cffi ‘SDL\_Event\*’ pointer.

**class** tcod.event.**WindowMoved** (*x: int, y: int*)

**type**

Always “WINDOWMOVED”.

**Type** str

**x**

Movement on the x-axis.

**Type** int

**y**

Movement on the y-axis.

**Type** int

**class** tcod.event.**WindowResized** (*type: str, width: int, height: int*)

**type**

“WINDOWRESIZED” or “WINDOWSIZECHANGED”

**Type** str

**width**

The current width of the window.

**Type** int

**height**

The current height of the window.

**Type** `int`

**class** `tcod.event.Undefined`

This class is a place holder for SDL events without their own `tcod.event` class.

**classmethod** `from_sdl_event` (*sdl\_event: Any*) → `tcod.event.Undefined`

Return a class instance from a python-cffi 'SDL\_Event\*' pointer.

**class** `tcod.event.EventDispatch`

This class dispatches events to methods depending on the events type attribute.

To use this class, make a sub-class and override the relevant `ev_*` methods. Then send events to the dispatch method.

Example:

```
import tcod
import tcod.event

class State(tcod.event.EventDispatch):
    def ev_quit(self, event):
        raise SystemExit()

    def ev_keydown(self, event):
        print(event)

    def ev_mousebuttondown(self, event):
        print(event)

    def ev_mousemotion(self, event):
        print(event)

root_console = tcod.console_init_root(80, 60)
state = State()
while True:
    for event in tcod.event.wait():
        state.dispatch(event)
```

**dispatch** (*event: Any*) → `None`

Send an event to an `ev_*` method.

\* will be the events type converted to lower-case.

If *event.type* is an empty string or `None` then it will be ignored.

**ev\_keydown** (*event: tcod.event.KeyDown*) → `None`

Called when a keyboard key is pressed or repeated.

**ev\_keyup** (*event: tcod.event.KeyUp*) → `None`

Called when a keyboard key is released.

**ev\_mousebuttondown** (*event: tcod.event.MouseButtonDown*) → `None`

Called when a mouse button is pressed.

**ev\_mousebuttonup** (*event: tcod.event.MouseButtonUp*) → `None`

Called when a mouse button is released.

**ev\_mousemotion** (*event: tcod.event.MouseMotion*) → `None`

Called when the mouse is moved.

- ev\_mousewheel** (*event: tcod.event.MouseWheel*) → None  
Called when the mouse wheel is scrolled.
- ev\_quit** (*event: tcod.event.Quit*) → None  
Called when the termination of the program is requested.
- ev\_textinput** (*event: tcod.event.TextInput*) → None  
Called to handle Unicode input.
- ev\_windowclose** (*event: tcod.event.WindowEvent*) → None  
Called when the window manager requests the window to be closed.
- ev\_windowenter** (*event: tcod.event.WindowEvent*) → None  
Called when the window gains mouse focus.
- ev\_windowexposed** (*event: tcod.event.WindowEvent*) → None  
Called when a window is exposed, and needs to be refreshed.  
This usually means a call to `tcod.console_flush` is necessary.
- ev\_windowfocusgained** (*event: tcod.event.WindowEvent*) → None  
Called when the window gains keyboard focus.
- ev\_windowfocuslost** (*event: tcod.event.WindowEvent*) → None  
Called when the window loses keyboard focus.
- ev\_windowhidden** (*event: tcod.event.WindowEvent*) → None  
Called when the window is hidden.
- ev\_windowleave** (*event: tcod.event.WindowEvent*) → None  
Called when the window loses mouse focus.
- ev\_windowmaximized** (*event: tcod.event.WindowEvent*) → None  
Called when the window is maximized.
- ev\_windowminimized** (*event: tcod.event.WindowEvent*) → None  
Called when the window is minimized.
- ev\_windowmoved** (*event: tcod.event.WindowMoved*) → None  
Called when the window is moved.
- ev\_windowresized** (*event: tcod.event.WindowResized*) → None  
Called when the window is resized.
- ev\_windowrestored** (*event: tcod.event.WindowEvent*) → None  
Called when the window is restored.
- ev\_windowshown** (*event: tcod.event.WindowEvent*) → None  
Called when the window is shown.
- ev\_windowsizechanged** (*event: tcod.event.WindowResized*) → None  
Called when the system or user changes the size of the window.

`tcod.event.get()` → Iterator[Any]  
Return an iterator for all pending events.

Events are processed as the iterator is consumed. Breaking out of, or discarding the iterator will leave the remaining events on the event queue.

Example:

```
for event in tcod.event.get():
    if event.type == "QUIT":
        print(event)
```

(continues on next page)

(continued from previous page)

```
        raise SystemExit()
    elif event.type == "KEYDOWN":
        print(event)
    elif event.type == "MOUSEBUTTONDOWN":
        print(event)
    elif event.type == "MOUSEMOTION":
        print(event)
    else:
        print(event)
```

`tcod.event.wait` (*timeout: Optional[float] = None*) → `Iterator[Any]`

Block until events exist, then return an event iterator.

*timeout* is the maximum number of seconds to wait as a floating point number with millisecond precision, or it can be `None` to wait forever.

Returns the same iterator as a call to `tcod.event.get`.

Example:

```
for event in tcod.event.wait():
    if event.type == "QUIT":
        print(event)
        raise SystemExit()
    elif event.type == "KEYDOWN":
        print(event)
    elif event.type == "MOUSEBUTTONDOWN":
        print(event)
    elif event.type == "MOUSEMOTION":
        print(event)
    else:
        print(event)
```



**class** `tcod.image.Image` (*width: int, height: int*)

**Parameters**

- **width** (*int*) – Width of the new Image.
- **height** (*int*) – Height of the new Image.

**width**

Read only width of this Image.

**Type** `int`

**height**

Read only height of this Image.

**Type** `int`

**blit** (*console: tcod.console.Console, x: float, y: float, bg\_blend: int, scale\_x: float, scale\_y: float, angle: float*) → `None`

Blit onto a Console using scaling and rotation.

**Parameters**

- **console** (`Console`) – Blit destination Console.
- **x** (*float*) – Console X position for the center of the Image blit.
- **y** (*float*) – Console Y position for the center of the Image blit. The Image blit is centered on this position.
- **bg\_blend** (*int*) – Background blending mode to use.
- **scale\_x** (*float*) – Scaling along Image x axis. Set to 1 for no scaling. Must be over 0.
- **scale\_y** (*float*) – Scaling along Image y axis. Set to 1 for no scaling. Must be over 0.
- **angle** (*float*) – Rotation angle in radians. (Clockwise?)

**blit\_2x** (*console: tcod.console.Console, dest\_x: int, dest\_y: int, img\_x: int = 0, img\_y: int = 0, img\_width: int = -1, img\_height: int = -1*) → None  
Blit onto a Console with double resolution.

#### Parameters

- **console** (*Console*) – Blit destination Console.
- **dest\_x** (*int*) – Console tile X position starting from the left at 0.
- **dest\_y** (*int*) – Console tile Y position starting from the top at 0.
- **img\_x** (*int*) – Left corner pixel of the Image to blit
- **img\_y** (*int*) – Top corner pixel of the Image to blit
- **img\_width** (*int*) – Width of the Image to blit. Use -1 for the full Image width.
- **img\_height** (*int*) – Height of the Image to blit. Use -1 for the full Image height.

**blit\_rect** (*console: tcod.console.Console, x: int, y: int, width: int, height: int, bg\_blend: int*) → None  
Blit onto a Console without scaling or rotation.

#### Parameters

- **console** (*Console*) – Blit destination Console.
- **x** (*int*) – Console tile X position starting from the left at 0.
- **y** (*int*) – Console tile Y position starting from the top at 0.
- **width** (*int*) – Use -1 for Image width.
- **height** (*int*) – Use -1 for Image height.
- **bg\_blend** (*int*) – Background blending mode to use.

**clear** (*color: Tuple[int, int, int]*) → None  
Fill this entire Image with color.

**Parameters** **color** (*Union[Tuple[int, int, int], Sequence[int]]*) – An (r, g, b) sequence or Color instance.

**get\_alpha** (*x: int, y: int*) → int  
Get the Image alpha of the pixel at x, y.

#### Parameters

- **x** (*int*) – X pixel of the image. Starting from the left at 0.
- **y** (*int*) – Y pixel of the image. Starting from the top at 0.

**Returns** The alpha value of the pixel. With 0 being fully transparent and 255 being fully opaque.

**Return type** *int*

**get\_mipmap\_pixel** (*left: float, top: float, right: float, bottom: float*) → Tuple[int, int, int]  
Get the average color of a rectangle in this Image.

Parameters should stay within the following limits: \* 0 <= left < right < Image.width \* 0 <= top < bottom < Image.height

#### Parameters

- **left** (*float*) – Left corner of the region.
- **top** (*float*) – Top corner of the region.

- **right** (*float*) – Right corner of the region.
- **bottom** (*float*) – Bottom corner of the region.

**Returns** An (r, g, b) tuple containing the averaged color value. Values are in a 0 to 255 range.

**Return type** Tuple[int, int, int]

**get\_pixel** (*x: int, y: int*) → Tuple[int, int, int]

Get the color of a pixel in this Image.

**Parameters**

- **x** (*int*) – X pixel of the Image. Starting from the left at 0.
- **y** (*int*) – Y pixel of the Image. Starting from the top at 0.

**Returns** An (r, g, b) tuple containing the pixels color value. Values are in a 0 to 255 range.

**Return type** Tuple[int, int, int]

**hflip** () → None

Horizontally flip this Image.

**invert** () → None

Invert all colors in this Image.

**put\_pixel** (*x: int, y: int, color: Tuple[int, int, int]*) → None

Change a pixel on this Image.

**Parameters**

- **x** (*int*) – X pixel of the Image. Starting from the left at 0.
- **y** (*int*) – Y pixel of the Image. Starting from the top at 0.
- **color** (*Union[Tuple[int, int, int], Sequence[int]]*) – An (r, g, b) sequence or Color instance.

**refresh\_console** (*console: tcod.console.Console*) → None

Update an Image created with `tcod.image_from_console`.

The console used with this function should have the same width and height as the Console given to `tcod.image_from_console`. The font width and height must also be the same as when `tcod.image_from_console` was called.

**Parameters console** (*Console*) – A Console with a pixel width and height matching this Image.

**rotate90** (*rotations: int = 1*) → None

Rotate this Image clockwise in 90 degree steps.

**Parameters rotations** (*int*) – Number of 90 degree clockwise rotations.

**save\_as** (*filename: str*) → None

Save the Image to a 32-bit .bmp or .png file.

**Parameters filename** (*Text*) – File path to same this Image.

**scale** (*width: int, height: int*) → None

Scale this Image to the new width and height.

**Parameters**

- **width** (*int*) – The new width of the Image after scaling.
- **height** (*int*) – The new height of the Image after scaling.

**set\_key\_color** (*color*: *Tuple[int, int, int]*) → None

Set a color to be transparent during blitting functions.

**Parameters** **color** (*Union[Tuple[int, int, int], Sequence[int]]*) – An (r, g, b) sequence or Color instance.

**vflip** () → None

Vertically flip this Image.

libtcod map attributes and field-of-view functions.

**class** `tcod.map.Map` (*width: int, height: int, order: str = 'C'*)

A map containing libtcod attributes.

Changed in version 4.1: *transparent*, *walkable*, and *fov* are now numpy boolean arrays.

Changed in version 4.3: Added *order* parameter.

#### Parameters

- **width** (*int*) – Width of the new Map.
- **height** (*int*) – Height of the new Map.
- **order** (*str*) – Which numpy memory order to use.

#### **width**

Read only width of this Map.

**Type** `int`

#### **height**

Read only height of this Map.

**Type** `int`

#### **transparent**

A boolean array of transparent cells.

#### **walkable**

A boolean array of walkable cells.

#### **fov**

A boolean array of the cells lit by `:any:compute_fov`.

Example:

```

>>> import tcod.map
>>> m = tcod.map.Map(width=3, height=4)
>>> m.walkable
array([[False, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False]]...)

# Like the rest of the tcod modules, all arrays here are
# in row-major order and are addressed with [y,x]
>>> m.transparent[:] = True # Sets all to True.
>>> m.transparent[1:3,0] = False # Sets (1, 0) and (2, 0) to False.
>>> m.transparent
array([[ True,  True,  True],
       [False,  True,  True],
       [False,  True,  True],
       [ True,  True,  True]]...)

>>> m.compute_fov(0, 0)
>>> m.fov
array([[ True,  True,  True],
       [ True,  True,  True],
       [False,  True,  True],
       [False, False,  True]]...)
>>> m.fov[3,1]
False

```

**compute\_fov** (*x*: int, *y*: int, *radius*: int = 0, *light\_walls*: bool = True, *algorithm*: int = 12) → None  
 Compute a field-of-view on the current instance.

#### Parameters

- **x** (*int*) – Point of view, x-coordinate.
- **y** (*int*) – Point of view, y-coordinate.
- **radius** (*int*) – Maximum view distance from the point of view.  
 A value of 0 will give an infinite distance.
- **light\_walls** (*bool*) – Light up walls, or only the floor.
- **algorithm** (*int*) – Defaults to `tcod.FOV_RESTRICTIVE`

If you already have transparency in a NumPy array then you could use `tcod.map_compute_fov` instead.

`tcod.map.compute_fov` (*transparency*: numpy.ndarray, *pov*: Tuple[int, int], *radius*: int = 0, *light\_walls*: bool = True, *algorithm*: int = 12) → numpy.ndarray

Return a boolean mask of the area covered by a field-of-view.

*transparency* is a 2 dimensional array where all non-zero values are considered transparent. The returned array will match the shape of this array.

*pov* is the point-of-view origin point. Areas are visible if they can be seen from this position. *pov* should be a 2D index matching the axes of the *transparency* array, and must be within the bounds of the *transparency* array.

*radius* is the maximum view distance from *pov*. If this is zero then the maximum distance is used.

If *light\_walls* is True then visible obstacles will be returned, otherwise only transparent areas will be.

*algorithm* is the field-of-view algorithm to run. The default value is `tcod.FOV_RESTRICTIVE`. The options are:

- `tcod.FOV_BASIC`: Simple ray-cast implementation.
- `tcod.FOV_DIAMOND`
- `tcod.FOV_SHADOW`: Recursive shadow caster.
- `tcod.FOV_PERMISSIVE(n)`: *n* starts at 0 (most restrictive) and goes up to 8 (most permissive.)
- `tcod.FOV_RESTRICTIVE`

New in version 9.3.

Changed in version 11.0: The parameters *x* and *y* have been changed to *pov*.

### Example

```
>>> explored = np.zeros((3, 5), dtype=bool, order="F")
>>> transparency = np.ones((3, 5), dtype=bool, order="F")
>>> transparency[:2, 2] = False
>>> transparency # Transparent area.
array([[ True,  True,  False,  True,  True],
       [ True,  True,  False,  True,  True],
       [ True,  True,  True,   True,  True]])
>>> visible = tcod.map.compute_fov(transparency, (0, 0))
>>> visible # Visible area.
array([[ True,  True,  True,  False,  False],
       [ True,  True,  True,  False,  False],
       [ True,  True,  True,  True,   False]])
>>> explored |= visible # Keep track of an explored area.
```

### See also:

`numpy.nonzero` `numpy.choose` `numpy.select`





The `Noise.sample_mgrid` and `Noise.sample_ogrid` methods are multi-threaded operations when the Python runtime supports OpenMP. Even when single threaded these methods will perform much better than multiple calls to `Noise.get_point`.

Example:

```
import numpy as np
import tcod
import tcod.noise

noise = tcod.noise.Noise(
    dimensions=2,
    algorithm=tcod.NOISE_SIMPLEX,
    implementation=tcod.noise.TURBULENCE,
    hurst=0.5,
    lacunarity=2.0,
    octaves=4,
    seed=None,
)

# Create a 5x5 open multi-dimensional mesh-grid.
ogrid = [np.arange(5, dtype=np.float32),
         np.arange(5, dtype=np.float32)]
print(ogrid)

# Scale the grid.
ogrid[0] *= 0.25
ogrid[1] *= 0.25

# Return the sampled noise from this grid of points.
samples = noise.sample_ogrid(ogrid)
print(samples)
```

```
class tcod.noise.Noise(dimensions: int, algorithm: int = 2, implementation: int = 0, hurst:  
float = 0.5, lacunarity: float = 2.0, octaves: float = 4, seed: Op-  
tional[tcod.random.Random] = None)
```

The `hurst` exponent describes the raggedness of the resultant noise, with a higher value leading to a smoother noise. Not used with `tcod.noise.SIMPLE`.

`lacunarity` is a multiplier that determines how fast the noise frequency increases for each successive octave. Not used with `tcod.noise.SIMPLE`.

#### Parameters

- **dimensions** (*int*) – Must be from 1 to 4.
- **algorithm** (*int*) – Defaults to `NOISE_SIMPLEX`
- **implementation** (*int*) – Defaults to `tcod.noise.SIMPLE`
- **hurst** (*float*) – The hurst exponent. Should be in the 0.0-1.0 range.
- **lacunarity** (*float*) – The noise lacunarity.
- **octaves** (*float*) – The level of detail on fBm and turbulence implementations.
- **seed** (*Optional[Random]*) – A `Random` instance, or `None`.

#### `noise_c`

A cffi pointer to a `TCOD_noise_t` object.

#### Type `CData`

**get\_point** (*x: float = 0, y: float = 0, z: float = 0, w: float = 0*) → `float`

Return the noise value at the (x, y, z, w) point.

#### Parameters

- **x** (*float*) – The position on the 1st axis.
- **y** (*float*) – The position on the 2nd axis.
- **z** (*float*) – The position on the 3rd axis.
- **w** (*float*) – The position on the 4th axis.

**sample\_mgrid** (*mgrid: numpy.ndarray*) → `numpy.ndarray`

Sample a mesh-grid array and return the result.

The `sample_ogrid` method performs better as there is a lot of overhead when working with large mesh-grids.

**Parameters** **mgrid** (*numpy.ndarray*) – A mesh-grid array of points to sample. A contiguous array of type `numpy.float32` is preferred.

#### Returns

An array of sampled points.

This array has the shape: `mgrid.shape[:-1]`. The `dtype` is `numpy.float32`.

**Return type** `numpy.ndarray`

**sample\_ogrid** (*ogrid: numpy.ndarray*) → `numpy.ndarray`

Sample an open mesh-grid array and return the result.

**Args** **ogrid** (`Sequence[Sequence[float]]`): An open mesh-grid.

#### Returns

An array of sampled points.

The `shape` is based on the lengths of the open mesh-grid arrays. The `dtype` is `numpy.float32`.

**Return type** `numpy.ndarray`



Example:

```
>>> import numpy as np
>>> import tcod.path
>>> dungeon = np.array(
...     [
...         [1, 0, 1, 1, 1],
...         [1, 0, 1, 0, 1],
...         [1, 1, 1, 0, 1],
...     ],
...     dtype=np.int8,
... )
...

# Create a pathfinder from a numpy array.
# This is the recommended way to use the tcod.path module.
>>> astar = tcod.path.AStar(dungeon)
>>> print(astar.get_path(0, 0, 2, 4))
[(1, 0), (2, 1), (1, 2), (0, 3), (1, 4), (2, 4)]
>>> astar.cost[0, 1] = 1 # You can access the map array via this attribute.
>>> print(astar.get_path(0, 0, 2, 4))
[(0, 1), (0, 2), (0, 3), (1, 4), (2, 4)]

# Create a pathfinder from an edge_cost function.
# Calling Python functions from C is known to be very slow.
>>> def edge_cost(my_x, my_y, dest_x, dest_y):
...     return dungeon[dest_x, dest_y]
...
>>> dijkstra = tcod.path.Dijkstra(
...     tcod.path.EdgeCostCallback(edge_cost, dungeon.shape),
... )
...
>>> dijkstra.set_goal(0, 0)
>>> print(dijkstra.get_path(2, 4))
[(0, 1), (0, 2), (0, 3), (1, 4), (2, 4)]
```

Changed in version 5.0: All path-finding functions now respect the NumPy array shape (if a NumPy array is used.)

```
class tcod.path.AStar (cost: Any, diagonal: float = 1.41)
```

**Parameters**

- **cost** (*Union[tcod.map.Map, numpy.ndarray, Any]*) –
- **diagonal** (*float*) – Multiplier for diagonal movement. A value of 0 will disable diagonal movement entirely.

```
get_path (start_x: int, start_y: int, goal_x: int, goal_y: int) → List[Tuple[int, int]]
```

Return a list of (x, y) steps to reach the goal point, if possible.

**Parameters**

- **start\_x** (*int*) – Starting X position.
- **start\_y** (*int*) – Starting Y position.
- **goal\_x** (*int*) – Destination X position.
- **goal\_y** (*int*) – Destination Y position.

**Returns** A list of points, or an empty list if there is no valid path.

**Return type** List[Tuple[int, int]]

```
class tcod.path.Dijkstra (cost: Any, diagonal: float = 1.41)
```

**Parameters**

- **cost** (*Union[tcod.map.Map, numpy.ndarray, Any]*) –
- **diagonal** (*float*) – Multiplier for diagonal movement. A value of 0 will disable diagonal movement entirely.

```
get_path (x: int, y: int) → List[Tuple[int, int]]
```

Return a list of (x, y) steps to reach the goal point, if possible.

```
set_goal (x: int, y: int) → None
```

Set the goal point and recompute the Dijkstra path-finder.

```
class tcod.path.EdgeCostCallback (callback: Callable[[int, int, int, int], float], shape: Tuple[int, int])
```

Calculate cost from an edge-cost callback.

*callback* is the custom userdata to send to the C call.

*shape* is a 2-item tuple representing the maximum boundary for the algorithm. The callback will not be called with parameters outside of these bounds.

Changed in version 5.0: Now only accepts a *shape* argument instead of *width* and *height*.

```
class tcod.path.NodeCostArray
```

Calculate cost from a numpy array of nodes.

*array* is a NumPy array holding the path-cost of each node. A cost of 0 means the node is blocking.

```
static __new__ (cls, array: numpy.ndarray) → tcod.path.NodeCostArray
```

Validate a numpy array and setup a C callback.

```
tcod.path.dijkstra2d (distance: numpy.array, cost: numpy.array, cardinal: Optional[int], diagonal: Optional[int]) → None
```

Return the computed distance of all nodes on a 2D Dijkstra grid.

*distance* is an input/output array of node distances. Is this often an array filled with maximum finite values and 1 or more points with a low value such as 0. Distance will flow from these low values to adjacent nodes based the cost to reach those nodes. This array is modified in-place.

*cost* is an array of node costs. Any node with a cost less than or equal to 0 is considered blocked off. Positive values are the distance needed to reach that node.

*cardinal* and *diagonal* are the cost multipliers for edges in those directions. A value of None or 0 will disable those directions. Typical values could be: 1, None, 1, 1, 2, 3, etc.

*out* is the output array and can be the same as *distance*.

## Example

```
>>> import numpy as np
>>> import tcod.path
>>> cost = np.ones((3, 3), dtype=np.uint8)
>>> cost[:2, 1] = 0
>>> cost
array([[1, 0, 1],
       [1, 0, 1],
       [1, 1, 1]], dtype=uint8)
>>> dist = tcod.path.maxarray((3, 3), dtype=np.int32)
>>> dist[0, 0] = 0
>>> dist
array([[0, 2147483647, 2147483647],
       [2147483647, 2147483647, 2147483647],
       [2147483647, 2147483647, 2147483647]]...)
>>> tcod.path.dijkstra2d(dist, cost, 2, 3)
>>> dist
array([[0, 2147483647, 10],
       [2, 2147483647, 8],
       [4, 5, 7]]...)
>>> path = tcod.path.hillclimb2d(dist, (2, 2), True, True)
>>> path
array([[2, 2],
       [2, 1],
       [1, 0],
       [0, 0]], dtype=int32)
>>> path = list(path[::-1])
>>> while path:
...     print(path.pop(0))
[0 0]
[1 0]
[2 1]
[2 2]
```

New in version 11.2.

`tcod.path.hillclimb2d` (*distance*: `numpy.array`, *start*: `Tuple[int, int]`, *cardinal*: `bool`, *diagonal*: `bool`)  
→ `numpy.array`

Return a path on a grid from *start* to the lowest point.

*distance* should be a fully computed distance array. This kind of array is returned by `dijkstra2d`.

*start* is a 2-item tuple with starting coordinates. The axes if these coordinates should match the axis of the *distance* array. An out-of-bounds *start* index will raise an `IndexError`.

At each step nodes adjacent to current will be checked for a value lower than the current one. Which directions

are checked is decided by the boolean values *cardinal* and *diagonal*. This process is repeated until all adjacent nodes are equal to or larger than the last point on the path.

The returned array is a 2D NumPy array with the shape: (length, axis). This array always includes both the starting and ending point and will always have at least one item.

Typical uses of the returned array will be to either convert it into a list which can be popped from, or transpose it and convert it into a tuple which can be used to index other arrays using NumPy's advanced indexing rules.

New in version 11.2.

`tcod.path.maxarray` (*shape*: *Tuple[int, ...]*, *dtype*: *Any = <class 'int'>*, *order*: *str = 'C'*) → `numpy.array`

Return a new array filled with the maximum finite value for *dtype*.

*shape* is of the new array. Same as other NumPy array initializers.

*dtype* should be a single NumPy integer type.

*order* can be "C" or "F".

This works the same as `np.full(shape, np.iinfo(dtype).max, dtype, order)`.

This kind of array is an ideal starting point for distance maps. Just set any point to a lower value such as 0 and then pass this array to a function such as *dijkstra2d*.



Usually it's recommend to the Python's standard library *random* module instead of this one.

However, you will need to use these generators to get deterministic results from the *Noise* and *BSP* classes.

**class** `tcod.random.Random` (*algorithm*: `int = 0`, *seed*: `Optional[Hashable] = None`)

The libtcod random number generator.

*algorithm* defaults to Mersenne Twister, it can be one of:

- `tcod.random.MERSENNE_TWISTER`
- `tcod.random.MULTIPLY_WITH_CARRY`

*seed* is a 32-bit number or any Python hashable object like a string. Using the same seed will cause the generator to return deterministic values. The default *seed* of `None` will generate a random seed instead.

**random\_c**

A cffi pointer to a `TCOD_random_t` object.

**Type** `CData`

Changed in version 9.1: Added `tcod.random.MULTIPLY_WITH_CARRY` constant. *algorithm* parameter now defaults to `tcod.random.MERSENNE_TWISTER`.

**\_\_getstate\_\_** () → `Any`

Pack the `self.random_c` attribute into a portable state.

**\_\_setstate\_\_** (*state*: `Any`) → `None`

Create a new `cdata` object with the stored paramaters.

**guass** (*mu*: `float`, *sigma*: `float`) → `float`

Return a random number using Gaussian distribution.

**Parameters**

- **mu** (`float`) – The median returned value.
- **sigma** (`float`) – The standard deviation.

**Returns** A random float.

**Return type** `float`

**inverse\_guass** (*mu: float, sigma: float*) → `float`

Return a random Gaussian number using the Box-Muller transform.

**Parameters**

- **mu** (*float*) – The median returned value.
- **sigma** (*float*) – The standard deviation.

**Returns** A random float.

**Return type** `float`

**randint** (*low: int, high: int*) → `int`

Return a random integer within the linear range:  $low \leq n \leq high$ .

**Parameters**

- **low** (*int*) – The lower bound of the random range.
- **high** (*int*) – The upper bound of the random range.

**Returns** A random integer.

**Return type** `int`

**uniform** (*low: float, high: float*) → `float`

Return a random floating number in the range:  $low \leq n \leq high$ .

**Parameters**

- **low** (*float*) – The lower bound of the random range.
- **high** (*float*) – The upper bound of the random range.

**Returns** A random float.

**Return type** `float`

Tileset and font related functions.

Remember to add the line `import tcod.tileset`, as importing this module is not implied by `import tcod`.

**class** `tcod.tileset.Tileset` (*tile\_width: int, tile\_height: int*)

A collection of graphical tiles.

This class is provisional, the API may change in the future.

**\_\_contains\_\_** (*codepoint: int*) → bool

Test if a tileset has a codepoint with *n* in tileset.

**get\_tile** (*codepoint: int*) → `numpy.ndarray`

Return a copy of a tile for the given codepoint.

If the tile does not exist yet then a blank array will be returned.

The tile will have a shape of (height, width, rgba) and a dtype of `uint8`. Note that most grey-scale tiles will only use the alpha channel and will usually have a solid white color channel.

**set\_tile** (*codepoint: int, tile: numpy.ndarray*) → None

Upload a tile into this array.

The tile can be in 32-bit color (height, width, rgba), or grey-scale (height, width). The tile should have a dtype of `np.uint8`.

This data may need to be sent to graphics card memory, this is a slow operation.

**tile\_height**

The height of the tile in pixels.

**tile\_shape**

The shape (height, width) of the tile in pixels.

**tile\_width**

The width of the tile in pixels.

`tcod.tileset.get_default` () → `tcod.tileset.Tileset`

Return a reference to the default Tileset.

This function is provisional. The API may change.

`tcod.tileset.load_truetype_font` (*path*: str, *tile\_width*: int, *tile\_height*: int) → `tcod.tileset.Tileset`  
Return a new `Tileset` from a `.ttf` or `.otf` file.

Same as `set_truetype_font`, but returns a `Tileset` instead. You can send this `Tileset` to `set_default`.

This function is provisional. The API may change.

`tcod.tileset.set_default` (*tileset*: `tcod.tileset.Tileset`) → None  
Set the default `tileset`.

The display will use this new `tileset` immediately.

This function only affects the `SDL2` and `OPENGL2` renderers.

This function is provisional. The API may change.

`tcod.tileset.set_truetype_font` (*path*: str, *tile\_width*: int, *tile\_height*: int) → None  
Set the default `tileset` from a `.ttf` or `.otf` file.

*path* is the file path for the font file.

*tile\_width* and *tile\_height* are the desired size of the tiles in the new `tileset`. The font will be scaled to fit the given *tile\_height* and *tile\_width*.

This function will only affect the `SDL2` and `OPENGL2` renderers.

This function must be called before `tcod.console_init_root`. Once the root console is setup you may call this function again to change the font. The `tileset` can be changed but the window will not be resized automatically.

New in version 9.2.

## 13.1 bsp

`tcod.bsp_new_with_size` (*x: int, y: int, w: int, h: int*) → `tcod.bsp.BSP`  
Create a new BSP instance with the given rectangle.

### Parameters

- **x** (*int*) – Rectangle left coordinate.
- **y** (*int*) – Rectangle top coordinate.
- **w** (*int*) – Rectangle width.
- **h** (*int*) – Rectangle height.

**Returns** A new BSP instance.

### Return type *BSP*

Deprecated since version 2.0: Call the *BSP* class instead.

`tcod.bsp_split_once` (*node: tcod.bsp.BSP, horizontal: bool, position: int*) → `None`  
Deprecated since version 2.0: Use *BSP.split\_once* instead.

`tcod.bsp_split_recursive` (*node: tcod.bsp.BSP, randomizer: Optional[tcod.random.Random], nb: int, minHSize: int, minVSize: int, maxHRatio: int, maxVRatio: int*) → `None`  
Deprecated since version 2.0: Use *BSP.split\_recursive* instead.

`tcod.bsp_resize` (*node: tcod.bsp.BSP, x: int, y: int, w: int, h: int*) → `None`  
Deprecated since version 2.0: Assign directly to *BSP* attributes instead.

`tcod.bsp_left` (*node: tcod.bsp.BSP*) → `Optional[tcod.bsp.BSP]`  
Deprecated since version 2.0: Use *BSP.children* instead.

`tcod.bsp_right` (*node: tcod.bsp.BSP*) → `Optional[tcod.bsp.BSP]`  
Deprecated since version 2.0: Use *BSP.children* instead.

`tcod.bsp_father` (*node: tcod.bsp.BSP*) → Optional[tcod.bsp.BSP]  
Deprecated since version 2.0: Use `BSP.parent` instead.

`tcod.bsp_is_leaf` (*node: tcod.bsp.BSP*) → bool  
Deprecated since version 2.0: Use `BSP.children` instead.

`tcod.bsp_contains` (*node: tcod.bsp.BSP, cx: int, cy: int*) → bool  
Deprecated since version 2.0: Use `BSP.contains` instead.

`tcod.bsp_find_node` (*node: tcod.bsp.BSP, cx: int, cy: int*) → Optional[tcod.bsp.BSP]  
Deprecated since version 2.0: Use `BSP.find_node` instead.

`tcod.bsp_traverse_pre_order` (*node: tcod.bsp.BSP, callback: Callable[[tcod.bsp.BSP, Any], None], userData: Any = 0*) → None  
Traverse this nodes hierarchy with a callback.  
Deprecated since version 2.0: Use `BSP.pre_order` instead.

`tcod.bsp_traverse_in_order` (*node: tcod.bsp.BSP, callback: Callable[[tcod.bsp.BSP, Any], None], userData: Any = 0*) → None  
Traverse this nodes hierarchy with a callback.  
Deprecated since version 2.0: Use `BSP.in_order` instead.

`tcod.bsp_traverse_post_order` (*node: tcod.bsp.BSP, callback: Callable[[tcod.bsp.BSP, Any], None], userData: Any = 0*) → None  
Traverse this nodes hierarchy with a callback.  
Deprecated since version 2.0: Use `BSP.post_order` instead.

`tcod.bsp_traverse_level_order` (*node: tcod.bsp.BSP, callback: Callable[[tcod.bsp.BSP, Any], None], userData: Any = 0*) → None  
Traverse this nodes hierarchy with a callback.  
Deprecated since version 2.0: Use `BSP.level_order` instead.

`tcod.bsp_traverse_inverted_level_order` (*node: tcod.bsp.BSP, callback: Callable[[tcod.bsp.BSP, Any], None], userData: Any = 0*) → None  
Traverse this nodes hierarchy with a callback.  
Deprecated since version 2.0: Use `BSP.inverted_level_order` instead.

`tcod.bsp_remove_sons` (*node: tcod.bsp.BSP*) → None  
Delete all children of a given node. Not recommended.

---

**Note:** This function will add unnecessary complexity to your code. Don't use it.

---

Deprecated since version 2.0: BSP deletion is automatic.

`tcod.bsp_delete` (*node: tcod.bsp.BSP*) → None  
Exists for backward compatibility. Does nothing.

BSP's created by this library are automatically garbage collected once there are no references to the tree. This function exists for backwards compatibility.

Deprecated since version 2.0: BSP deletion is automatic.

## 13.2 color

**class** `tcod.Color` (*r: int = 0, g: int = 0, b: int = 0*)

**Parameters**

- **r** (*int*) – Red value, from 0 to 255.
- **g** (*int*) – Green value, from 0 to 255.
- **b** (*int*) – Blue value, from 0 to 255.

**r**

Red value, always normalised to 0-255.

Deprecated since version 9.2: Color attributes will not be mutable in the future.

**Type** *int***g**

Green value, always normalised to 0-255.

Deprecated since version 9.2: Color attributes will not be mutable in the future.

**Type** *int***b**

Blue value, always normalised to 0-255.

Deprecated since version 9.2: Color attributes will not be mutable in the future.

**Type** *int***\_\_getitem\_\_** (*index: Any*) → *Any*

Deprecated since version 9.2: Accessing colors via a letter index is deprecated.

**\_\_eq\_\_** (*other: Any*) → *bool*

Compare equality between colors.

Also compares with standard sequences such as 3-item tuples or lists.

**\_\_repr\_\_** () → *str*

Return a printable representation of the current color.

`tcod.color.lerp` (*c1: Tuple[int, int, int], c2: Tuple[int, int, int], a: float*) → `tcod.color.Color`

Return the linear interpolation between two colors.

*a* is the interpolation value, with 0 returning *c1*, 1 returning *c2*, and 0.5 returning a color halfway between both.**Parameters**

- **c1** (*Union[Tuple[int, int, int], Sequence[int]]*) – The first color. At *a=0*.
- **c2** (*Union[Tuple[int, int, int], Sequence[int]]*) – The second color. At *a=1*.
- **a** (*float*) – The interpolation value,

**Returns** The interpolated Color.**Return type** *Color*`tcod.color.set_hsv` (*c: tcod.color.Color, h: float, s: float, v: float*) → *None*

Set a color using: hue, saturation, and value parameters.

Does not return a new Color. *c* is modified inplace.**Parameters**

- **c** (*Union[Color, List[Any]]*) – A Color instance, or a list of any kind.

- **h** (*float*) – Hue, from 0 to 360.
- **s** (*float*) – Saturation, from 0 to 1.
- **v** (*float*) – Value, from 0 to 1.

`tcod.color_get_hsv` (*c: Tuple[int, int, int]*) → `Tuple[float, float, float]`

Return the (hue, saturation, value) of a color.

**Parameters** *c* (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or `Color` instance.

**Returns** A tuple with (hue, saturation, value) values, from 0 to 1.

**Return type** `Tuple[float, float, float]`

`tcod.color_scale_HSV` (*c: tcod.color.Color, scoef: float, vcoef: float*) → `None`

Scale a color's saturation and value.

Does not return a new `Color`. *c* is modified inplace.

#### Parameters

- **c** (`Union[Color, List[int]]`) – A `Color` instance, or an [r, g, b] list.
- **scoef** (*float*) – Saturation multiplier, from 0 to 1. Use 1 to keep current saturation.
- **vcoef** (*float*) – Value multiplier, from 0 to 1. Use 1 to keep current value.

`tcod.color_gen_map` (*colors: Iterable[Tuple[int, int, int]], indexes: Iterable[int]*) → `List[tcod.color.Color]`

Return a smoothly defined scale of colors.

If *indexes* is [0, 3, 9] for example, the first color from *colors* will be returned at 0, the 2nd will be at 3, and the 3rd will be at 9. All in-betweens will be filled with a gradient.

#### Parameters

- **colors** (`Iterable[Union[Tuple[int, int, int], Sequence[int]]]`) – Array of colors to be sampled.
- **indexes** (`Iterable[int]`) – A list of indexes.

**Returns** A list of `Color` instances.

**Return type** `List[Color]`

### Example

```
>>> tcod.color_gen_map([(0, 0, 0), (255, 128, 0)], [0, 5])
[Color(0, 0, 0), Color(51, 25, 0), Color(102, 51, 0), Color(153, 76, 0),
↪Color(204, 102, 0), Color(255, 128, 0)]
```

## 13.2.1 color controls

Configurable color control constants which can be set up with `tcod.console_set_color_control`.

`tcod.COLCTRL_1`

`tcod.COLCTRL_2`

`tcod.COLCTRL_3`

`tcod.COLCTRL_4`



`tcod.COLCTRL_5``tcod.COLCTRL_STOP``tcod.COLCTRL_FORE_RGB``tcod.COLCTRL_BACK_RGB`

## 13.3 console

`tcod.console_set_custom_font` (*fontFile*: AnyStr, *flags*: int = 1, *nb\_char\_horiz*: int = 0, *nb\_char\_vertic*: int = 0) → None

Load the custom font file at *fontFile*.

Call this before function before calling `tcod.console_init_root`.

Flags can be a mix of the following:

- `tcod.FONT_LAYOUT_ASCII_INCOL`: Decode tileset raw in column-major order.
- `tcod.FONT_LAYOUT_ASCII_INROW`: Decode tileset raw in row-major order.
- `tcod.FONT_TYPE_GREYSCALE`: Force tileset to be read as greyscale.
- `tcod.FONT_TYPE_GRAYSCALE`
- `tcod.FONT_LAYOUT_TCOD`: Unique layout used by libtcod.
- `tcod.FONT_LAYOUT_CP437`: Decode a row-major Code Page 437 tileset into Unicode.

*nb\_char\_horiz* and *nb\_char\_vertic* are the columns and rows of the font file respectfully.

`tcod.console_init_root` (*w*: int, *h*: int, *title*: Optional[str] = None, *fullscreen*: bool = False, *renderer*: Optional[int] = None, *order*: str = 'C', *vsync*: Optional[bool] = None) → `tcod.console.Console`

Set up the primary display and return the root console.

*w* and *h* are the columns and rows of the new window (in tiles.)

*title* is an optional string to display on the windows title bar.

*fullscreen* determines if the window will start in fullscreen. Fullscreen mode is unreliable unless the renderer is set to `tcod.RENDERER_SDL2` or `tcod.RENDERER_OPENGL2`.

*renderer* is the rendering back-end that libtcod will use. If you don't know which to pick, then use `tcod.RENDERER_SDL2`. Options are:

- `tcod.RENDERER_SDL`: A deprecated software/SDL2 renderer.
- `tcod.RENDERER_OPENGL`: A deprecated SDL2/OpenGL1 renderer.
- `tcod.RENDERER_GLSL`: A deprecated SDL2/OpenGL2 renderer.
- `tcod.RENDERER_SDL2`: The recommended SDL2 renderer. Rendering is decided by SDL2 and can be changed by using an SDL2 hint.
- `tcod.RENDERER_OPENGL2`: An SDL2/OpenGL2 renderer. Usually faster than regular SDL2. Requires OpenGL 2.0 Core.

*order* will affect how the array attributes of the returned root console are indexed. *order*='C' is the default, but *order*='F' is recommended.

If *vsync* is True then the frame-rate will be synchronized to the monitors vertical refresh rate. This prevents screen tearing and avoids wasting computing power on overdraw. If *vsync* is False then the frame-rate will be

uncapped. The default is False but will change to True in the future. This option only works with the SDL2 or OPENG2 renderers, any other renderer will always have *vsync* disabled.

Changed in version 4.3: Added *order* parameter. *title* parameter is now optional.

Changed in version 8.0: The default *renderer* is now automatic instead of always being *RENDERER\_SDL*.

Changed in version 10.1: Added the *vsync* parameter.

`tcod.console_flush()` → None

Update the display to represent the root console's current state.

`tcod.console_blit(src: tcod.console.Console, x: int, y: int, w: int, h: int, dst: tcod.console.Console, xdst: int, ydst: int, ffade: float = 1.0, bfade: float = 1.0)` → None

Blit the console *src* from *x,y,w,h* to console *dst* at *xdst,ydst*.

Deprecated since version 8.5: Call the `Console.blit` method instead.

`tcod.console_check_for_keypress(flags: int = 2)` → `tcod.libtcodpy.Key`

Deprecated since version 9.3: Use the `tcod.event.get` function to check for events.

`tcod.console_clear(con: tcod.console.Console)` → None

Reset a console to its default colors and the space character.

**Parameters** *con* (`Console`) – Any Console instance.

**See also:**

`console_set_default_background console_set_default_foreground`

Deprecated since version 8.5: Call the `Console.clear` method instead.

`tcod.console_credits()` → None

`tcod.console_credits_render(x: int, y: int, alpha: bool)` → bool

`tcod.console_credits_reset()` → None

`tcod.console_delete(con: tcod.console.Console)` → None

Closes the window if *con* is the root console.

libtcod objects are automatically garbage collected once they go out of scope.

This function exists for backwards compatibility.

Deprecated since version 9.3: This function is not needed for normal `tcod.console.Console`'s. The root console should be used in a with statement instead to ensure that it closes.

`tcod.console_fill_background(con: tcod.console.Console, r: Sequence[int], g: Sequence[int], b: Sequence[int])` → None

Fill the background of a console with *r,g,b*.

**Parameters**

- *con* (`Console`) – Any Console instance.
- *r* (`Sequence[int]`) – An array of integers with a length of width\*height.
- *g* (`Sequence[int]`) – An array of integers with a length of width\*height.
- *b* (`Sequence[int]`) – An array of integers with a length of width\*height.

Deprecated since version 8.4: You should assign to `tcod.console.Console.bg` instead.

`tcod.console_fill_char(con: tcod.console.Console, arr: Sequence[int])` → None

Fill the character tiles of a console with an array.

*arr* is an array of integers with a length of the console's width and height.

Deprecated since version 8.4: You should assign to `tcod.console.Console.ch` instead.

`tcod.console_fill_foreground` (*con*: `tcod.console.Console`, *r*: `Sequence[int]`, *g*: `Sequence[int]`, *b*: `Sequence[int]`) → `None`

Fill the foreground of a console with r,g,b.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **r** (`Sequence[int]`) – An array of integers with a length of width\*height.
- **g** (`Sequence[int]`) – An array of integers with a length of width\*height.
- **b** (`Sequence[int]`) – An array of integers with a length of width\*height.

Deprecated since version 8.4: You should assign to `tcod.console.Console.fg` instead.

`tcod.console_from_file` (*filename*: `str`) → `tcod.console.Console`

Return a new console object from a filename.

The file format is automatically determined. This can load REXPaint `.xp`, ASCII Paint `.apf`, or Non-delimited ASCII `.asc` files.

**Parameters** **filename** (`Text`) – The path to the file, as a string.

Returns: A new `any'Console'` instance.

`tcod.console_from_xp` (*filename*: `str`) → `tcod.console.Console`

Return a single console from a REXPaint `.xp` file.

`tcod.console_get_alignment` (*con*: `tcod.console.Console`) → `int`

Return this console's current alignment mode.

**Parameters** **con** (`Console`) – Any Console instance.

Deprecated since version 8.5: Check `Console.default_alignment` instead.

`tcod.console_get_background_flag` (*con*: `tcod.console.Console`) → `int`

Return this console's current blend mode.

**Parameters** **con** (`Console`) – Any Console instance.

Deprecated since version 8.5: Check `Console.default_bg_blend` instead.

`tcod.console_get_char` (*con*: `tcod.console.Console`, *x*: `int`, *y*: `int`) → `int`

Return the character at the x,y of this console.

Deprecated since version 8.4: Array access performs significantly faster than using this function. See `Console.ch`.

`tcod.console_get_char_background` (*con*: `tcod.console.Console`, *x*: `int`, *y*: `int`) → `tcod.color.Color`

Return the background color at the x,y of this console.

Deprecated since version 8.4: Array access performs significantly faster than using this function. See `Console.bg`.

`tcod.console_get_char_foreground` (*con*: `tcod.console.Console`, *x*: `int`, *y*: `int`) → `tcod.color.Color`

Return the foreground color at the x,y of this console.

Deprecated since version 8.4: Array access performs significantly faster than using this function. See `Console.fg`.

`tcod.console_get_default_background` (*con*: `tcod.console.Console`) → `tcod.color.Color`

Return this console's default background color.

Deprecated since version 8.5: Use `Console.default_bg` instead.

`tcod.console_get_default_foreground` (*con: tcod.console.Console*) → `tcod.color.Color`  
Return this consoles default foreground color.

Deprecated since version 8.5: Use `Console.default_fg` instead.

`tcod.console_get_fade` () → `int`

`tcod.console_get_fading_color` () → `tcod.color.Color`

`tcod.console_get_height` (*con: tcod.console.Console*) → `int`  
Return the height of a console.

**Parameters** `con` (`Console`) – Any Console instance.

**Returns** The height of a Console.

**Return type** `int`

Deprecated since version 2.0: Use `Console.height` instead.

`tcod.console_get_height_rect` (*con: tcod.console.Console, x: int, y: int, w: int, h: int, fmt: str*) → `int`  
Return the height of this text once word-wrapped into this rectangle.

**Returns** The number of lines of text once word-wrapped.

**Return type** `int`

Deprecated since version 8.5: Use `Console.get_height_rect` instead.

`tcod.console_get_width` (*con: tcod.console.Console*) → `int`  
Return the width of a console.

**Parameters** `con` (`Console`) – Any Console instance.

**Returns** The width of a Console.

**Return type** `int`

Deprecated since version 2.0: Use `Console.width` instead.

`tcod.console_hline` (*con: tcod.console.Console, x: int, y: int, l: int, flag: int = 13*) → `None`  
Draw a horizontal line on the console.

This always uses the character 196, the horizontal line character.

Deprecated since version 8.5: Use `Console.hline` instead.

`tcod.console_is_fullscreen` () → `bool`  
Returns True if the display is fullscreen.

**Returns** True if the display is fullscreen, otherwise False.

**Return type** `bool`

`tcod.console_is_key_pressed` (*key: int*) → `bool`

`tcod.console_is_window_closed` () → `bool`

Returns True if the window has received and exit event.

Deprecated since version 9.3: Use the `tcod.event` module to check for “QUIT” type events.

`tcod.console_load_apf` (*con: tcod.console.Console, filename: str*) → `bool`  
Update a console from an ASCII Paint `.apf` file.

`tcod.console_load_asc` (*con: tcod.console.Console, filename: str*) → `bool`  
Update a console from a non-delimited ASCII `.asc` file.

`tcod.console_load_xp` (*con: tcod.console.Console, filename: str*) → bool  
Update a console from a REXPaint *.xp* file.

`tcod.console_list_load_xp` (*filename: str*) → Optional[List[tcod.console.Console]]  
Return a list of consoles from a REXPaint *.xp* file.

`tcod.console_list_save_xp` (*console\_list: Sequence[tcod.console.Console], filename: str, compress\_level: int = 9*) → bool  
Save a list of consoles to a REXPaint *.xp* file.

`tcod.console_map_ascii_code_to_font` (*asciiCode: int, fontCharX: int, fontCharY: int*) → None  
Set a character code to new coordinates on the tile-set.

*asciiCode* must be within the bounds created during the initialization of the loaded tile-set. For example, you can't use 255 here unless you have a 256 tile tile-set loaded. This applies to all functions in this group.

#### Parameters

- **asciiCode** (*int*) – The character code to change.
- **fontCharX** (*int*) – The X tile coordinate on the loaded tileset. 0 is the leftmost tile.
- **fontCharY** (*int*) – The Y tile coordinate on the loaded tileset. 0 is the topmost tile.

`tcod.console_map_ascii_codes_to_font` (*firstAsciiCode: int, nbCodes: int, fontCharX: int, fontCharY: int*) → None  
Remap a contiguous set of codes to a contiguous set of tiles.

Both the tile-set and character codes must be contiguous to use this function. If this is not the case you may want to use `console_map_ascii_code_to_font`.

#### Parameters

- **firstAsciiCode** (*int*) – The starting character code.
- **nbCodes** (*int*) – The length of the contiguous set.
- **fontCharX** (*int*) – The starting X tile coordinate on the loaded tileset. 0 is the leftmost tile.
- **fontCharY** (*int*) – The starting Y tile coordinate on the loaded tileset. 0 is the topmost tile.

`tcod.console_map_string_to_font` (*s: str, fontCharX: int, fontCharY: int*) → None  
Remap a string of codes to a contiguous set of tiles.

#### Parameters

- **s** (*AnyStr*) – A string of character codes to map to new values. The null character `'x00'` will prematurely end this function.
- **fontCharX** (*int*) – The starting X tile coordinate on the loaded tileset. 0 is the leftmost tile.
- **fontCharY** (*int*) – The starting Y tile coordinate on the loaded tileset. 0 is the topmost tile.

`tcod.console_new` (*w: int, h: int*) → `tcod.console.Console`  
Return an offscreen console of size: w,h.

Deprecated since version 8.5: Create new consoles using `tcod.console.Console` instead of this function.

`tcod.console_print` (*con: tcod.console.Console, x: int, y: int, fmt: str*) → None  
Print a color formatted string on a console.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (`int`) – Character x position from the left.
- **y** (`int`) – Character y position from the top.
- **fmt** (`AnyStr`) – A unicode or bytes string optionally using color codes.

Deprecated since version 8.5: Use `Console.print_` instead.

`tcod.console_print_ex` (*con: tcod.console.Console, x: int, y: int, flag: int, alignment: int, fmt: str*) → `None`  
Print a string on a console using a blend mode and alignment mode.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (`int`) – Character x position from the left.
- **y** (`int`) – Character y position from the top.

Deprecated since version 8.5: Use `Console.print_` instead.

`tcod.console_print_frame` (*con: tcod.console.Console, x: int, y: int, w: int, h: int, clear: bool = True, flag: int = 13, fmt: str = ""*) → `None`  
Draw a framed rectangle with optional text.

This uses the default background color and blend mode to fill the rectangle and the default foreground to draw the outline.

*fmt* will be printed on the inside of the rectangle, word-wrapped. If *fmt* is empty then no title will be drawn.

Changed in version 8.2: Now supports Unicode strings.

Deprecated since version 8.5: Use `Console.print_frame` instead.

`tcod.console_print_rect` (*con: tcod.console.Console, x: int, y: int, w: int, h: int, fmt: str*) → `int`  
Print a string constrained to a rectangle.

If  $h > 0$  and the bottom of the rectangle is reached, the string is truncated. If  $h = 0$ , the string is only truncated if it reaches the bottom of the console.

**Returns** The number of lines of text once word-wrapped.

**Return type** `int`

Deprecated since version 8.5: Use `Console.print_rect` instead.

`tcod.console_print_rect_ex` (*con: tcod.console.Console, x: int, y: int, w: int, h: int, flag: int, alignment: int, fmt: str*) → `int`  
Print a string constrained to a rectangle with blend and alignment.

**Returns** The number of lines of text once word-wrapped.

**Return type** `int`

Deprecated since version 8.5: Use `Console.print_rect` instead.

`tcod.console_put_char` (*con: tcod.console.Console, x: int, y: int, c: Union[int, str], flag: int = 13*) → `None`  
Draw the character *c* at *x,y* using the default colors and a blend mode.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (`int`) – Character x position from the left.

- **y** (*int*) – Character y position from the top.
- **c** (*Union[int, AnyStr]*) – Character to draw, can be an integer or string.
- **flag** (*int*) – Blending mode to use, defaults to BKGND\_DEFAULT.

`tcod.console_put_char_ex` (*con: tcod.console.Console, x: int, y: int, c: Union[int, str], fore: Tuple[int, int, int], back: Tuple[int, int, int]*) → None  
 Draw the character `c` at `x,y` using the colors `fore` and `back`.

#### Parameters

- **con** (*Console*) – Any Console instance.
- **x** (*int*) – Character x position from the left.
- **y** (*int*) – Character y position from the top.
- **c** (*Union[int, AnyStr]*) – Character to draw, can be an integer or string.
- **fore** (*Union[Tuple[int, int, int], Sequence[int]]*) – An (r, g, b) sequence or Color instance.
- **back** (*Union[Tuple[int, int, int], Sequence[int]]*) – An (r, g, b) sequence or Color instance.

`tcod.console_rect` (*con: tcod.console.Console, x: int, y: int, w: int, h: int, clr: bool, flag: int = 13*) → None  
 Draw a the background color on a rect optionally clearing the text.

If `clr` is True the affected tiles are changed to space character.

Deprecated since version 8.5: Use `Console.rect` instead.

`tcod.console_save_apf` (*con: tcod.console.Console, filename: str*) → bool  
 Save a console to an ASCII Paint `.apf` file.

`tcod.console_save_asc` (*con: tcod.console.Console, filename: str*) → bool  
 Save a console to a non-delimited ASCII `.asc` file.

`tcod.console_save_xp` (*con: tcod.console.Console, filename: str, compress\_level: int = 9*) → bool  
 Save a console to a REXPaint `.xp` file.

`tcod.console_set_alignment` (*con: tcod.console.Console, alignment: int*) → None  
 Change this consoles current alignment mode.

- `tcod.LEFT`
- `tcod.CENTER`
- `tcod.RIGHT`

#### Parameters

- **con** (*Console*) – Any Console instance.
- **alignment** (*int*) –

Deprecated since version 8.5: Set `Console.default_alignment` instead.

`tcod.console_set_background_flag` (*con: tcod.console.Console, flag: int*) → None  
 Change the default blend mode for this console.

#### Parameters

- **con** (*Console*) – Any Console instance.

- **flag** (*int*) – Blend mode to use by default.

Deprecated since version 8.5: Set `Console.default_bg_blend` instead.

`tcod.console_set_char` (*con: tcod.console.Console, x: int, y: int, c: Union[int, str]*) → None

Change the character at x,y to c, keeping the current colors.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (*int*) – Character x position from the left.
- **y** (*int*) – Character y position from the top.
- **c** (`Union[int, AnyStr]`) – Character to draw, can be an integer or string.

Deprecated since version 8.4: Array access performs significantly faster than using this function. See `Console.ch`.

`tcod.console_set_char_background` (*con: tcod.console.Console, x: int, y: int, col: Tuple[int, int, int], flag: int = 1*) → None

Change the background color of x,y to col using a blend mode.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (*int*) – Character x position from the left.
- **y** (*int*) – Character y position from the top.
- **col** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.
- **flag** (*int*) – Blending mode to use, defaults to BKGND\_SET.

`tcod.console_set_char_foreground` (*con: tcod.console.Console, x: int, y: int, col: Tuple[int, int, int]*) → None

Change the foreground color of x,y to col.

#### Parameters

- **con** (`Console`) – Any Console instance.
- **x** (*int*) – Character x position from the left.
- **y** (*int*) – Character y position from the top.
- **col** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.

Deprecated since version 8.4: Array access performs significantly faster than using this function. See `Console.fg`.

`tcod.console_set_color_control` (*con: int, fore: Tuple[int, int, int], back: Tuple[int, int, int]*) → None

Configure color controls.

#### Parameters

- **con** (*int*) – Color control constant to modify.
- **fore** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.
- **back** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.



`tcod.console_set_default_background` (*con*: `tcod.console.Console`, *col*: `Tuple[int, int, int]`) → None  
Change the default background color for a console.

**Parameters**

- **con** (`Console`) – Any Console instance.
- **col** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.

Deprecated since version 8.5: Use `Console.default_bg` instead.

`tcod.console_set_default_foreground` (*con*: `tcod.console.Console`, *col*: `Tuple[int, int, int]`) → None  
Change the default foreground color for a console.

**Parameters**

- **con** (`Console`) – Any Console instance.
- **col** (`Union[Tuple[int, int, int], Sequence[int]]`) – An (r, g, b) sequence or Color instance.

Deprecated since version 8.5: Use `Console.default_fg` instead.

`tcod.console_set_fade` (*fade*: `int`, *fadingColor*: `Tuple[int, int, int]`) → None

`tcod.console_set_fullscreen` (*fullscreen*: `bool`) → None  
Change the display to be fullscreen or windowed.

**Parameters** **fullscreen** (`bool`) – Use True to change to fullscreen. Use False to change to windowed.

`tcod.console_set_key_color` (*con*: `tcod.console.Console`, *col*: `Tuple[int, int, int]`) → None  
Set a consoles blit transparent color.

Deprecated since version 8.5: Pass the key color to `tcod.console.Console.blit` instead of calling this function.

`tcod.console_set_window_title` (*title*: `str`) → None  
Change the current title bar string.

**Parameters** **title** (`AnyStr`) – A string to change the title bar to.

`tcod.console_vline` (*con*: `tcod.console.Console`, *x*: `int`, *y*: `int`, *l*: `int`, *flag*: `int = 13`) → None  
Draw a vertical line on the console.

This always uses the character 179, the vertical line character.

Deprecated since version 8.5: Use `Console.vline` instead.

`tcod.console_wait_for_keypress` (*flush*: `bool`) → `tcod.libtcodpy.Key`  
Block until the user presses a key, then returns a new Key.

**Parameters** **bool** (*flush*) – If True then the event queue is cleared before waiting for the next event.

**Returns** A new Key instance.

**Return type** `Key`

Deprecated since version 9.3: Use the `tcod.event.wait` function to wait for events.

## 13.4 Event

```
class tcod.Key
    Key Event instance

    vk
        TCOD_keycode_t key code
        Type int

    c
        character if vk == TCODK_CHAR else 0
        Type int

    text
        text[TCOD_KEY_TEXT_SIZE]; text if vk == TCODK_TEXT else text[0] == ""
        Type Text

    pressed
        does this correspond to a key press or key release event?
        Type bool

    lalt
        True when left alt is held.
        Type bool

    lctrl
        True when left control is held.
        Type bool

    lmeta
        True when left meta key is held.
        Type bool

    ralt
        True when right alt is held.
        Type bool

    rctrl
        True when right control is held.
        Type bool

    rmeta
        True when right meta key is held.
        Type bool

    shift
        True when any shift is held.
        Type bool

    Deprecated since version 9.3: Use events from the tcod.event module instead.

    __repr__ () → str
        Return a representation of this Key object.
```

```
class tcod.Mouse
    Mouse event instance

    x
        Absolute mouse position at pixel x.
        Type int

    y
        Type int

    dx
        Movement since last update in pixels.
        Type int

    dy
        Type int

    cx
        Cell coordinates in the root console.
        Type int

    cy
        Type int

    dcx
        Movement since last update in console cells.
        Type int

    dcy
        Type int

    lbutton
        Left button status.
        Type bool

    rbutton
        Right button status.
        Type bool

    mbutton
        Middle button status.
        Type bool

    lbutton_pressed
        Left button pressed event.
        Type bool

    rbutton_pressed
        Right button pressed event.
        Type bool

    mbutton_pressed
        Middle button pressed event.
        Type bool
```

**wheel\_up**

Wheel up event.

Type `bool`

**wheel\_down**

Wheel down event.

Type `bool`

Deprecated since version 9.3: Use events from the `tcod.event` module instead.

`__repr__` () → str

Return a representation of this Mouse object.

### 13.4.1 Event Types

`tcod.EVENT_NONE`

`tcod.EVENT_KEY_PRESS`

`tcod.EVENT_KEY_RELEASE`

`tcod.EVENT_KEY`

Same as `tcod.EVENT_KEY_PRESS` | `tcod.EVENT_KEY_RELEASE`

`tcod.EVENT_MOUSE_MOVE`

`tcod.EVENT_MOUSE_PRESS`

`tcod.EVENT_MOUSE_RELEASE`

`tcod.EVENT_MOUSE`

Same as `tcod.EVENT_MOUSE_MOVE` | `tcod.EVENT_MOUSE_PRESS` | `tcod.EVENT_MOUSE_RELEASE`

`tcod.EVENT_FINGER_MOVE`

`tcod.EVENT_FINGER_PRESS`

`tcod.EVENT_FINGER_RELEASE`

`tcod.EVENT_FINGER`

Same as `tcod.EVENT_FINGER_MOVE` | `tcod.EVENT_FINGER_PRESS` | `tcod.EVENT_FINGER_RELEASE`

`tcod.EVENT_ANY`

Same as `tcod.EVENT_KEY` | `tcod.EVENT_MOUSE` | `tcod.EVENT_FINGER`

## 13.5 sys

`tcod.sys_set_fps` (*fps*: int) → None

Set the maximum frame rate.

You can disable the frame limit again by setting *fps* to 0.

**Parameters** *fps* (int) – A frame rate limit (i.e. 60)

`tcod.sys_get_fps` () → int

Return the current frames per second.

This the actual frame rate, not the frame limit set by `tcod.sys_set_fps`.

This number is updated every second.

**Returns** The currently measured frame rate.

**Return type** `int`

`tcod.sys_get_last_frame_length()` → float  
Return the delta time of the last rendered frame in seconds.

**Returns** The delta time of the last rendered frame.

**Return type** `float`

`tcod.sys_sleep_milli(val: int)` → None  
Sleep for 'val' milliseconds.

**Parameters** `val (int)` – Time to sleep for in milliseconds.

Deprecated since version 2.0: Use `time.sleep` instead.

`tcod.sys_elapsed_milli()` → int  
Get number of milliseconds since the start of the program.

**Returns** Time since the progeam has started in milliseconds.

**Return type** `int`

Deprecated since version 2.0: Use `time.clock` instead.

`tcod.sys_elapsed_seconds()` → float  
Get number of seconds since the start of the program.

**Returns** Time since the progeam has started in seconds.

**Return type** `float`

Deprecated since version 2.0: Use `time.clock` instead.

`tcod.sys_set_renderer(renderer: int)` → None  
Change the current rendering mode to `renderer`.

Deprecated since version 2.0: `RENDERER_GLSL` and `RENDERER_OPENGL` are not currently available.

`tcod.sys_get_renderer()` → int  
Return the current rendering mode.

`tcod.sys_save_screenshot(name: Optional[str] = None)` → None  
Save a screenshot to a file.

By default this will automatically save screenshots in the working directory.

The automatic names are formatted as `screenshotNNN.png`. For example: `screenshot000.png`, `screenshot001.png`, etc. Whichever is available first.

**Parameters** `Optional[AnyStr] (file)` – File path to save screenshot.

`tcod.sys_force_fullscreen_resolution(width: int, height: int)` → None  
Force a specific resolution in fullscreen.

Will use the smallest available resolution so that:

- resolution width  $\geq$  width and resolution width  $\geq$  root console width \* font char width
- resolution height  $\geq$  height and resolution height  $\geq$  root console height \* font char height

**Parameters**

- `width (int)` – The desired resolution width.

- **height** (*int*) – The desired resolution height.

`tcod.sys_get_current_resolution()` → `Tuple[int, int]`  
Return the current resolution as (width, height)

**Returns** The current resolution.

**Return type** `Tuple[int, int]`

`tcod.sys_get_char_size()` → `Tuple[int, int]`  
Return the current fonts character size as (width, height)

**Returns** The current font glyph size in (width, height)

**Return type** `Tuple[int, int]`

`tcod.sys_update_char(asciiCode: int, fontx: int, fonty: int, img: tcod.image.Image, x: int, y: int)` → `None`  
Dynamically update the current font with img.

All cells using this `asciiCode` will be updated at the next call to `tcod.console_flush`.

#### Parameters

- **asciiCode** (*int*) – Ascii code corresponding to the character to update.
- **fontx** (*int*) – Left coordinate of the character in the bitmap font (in tiles)
- **fonty** (*int*) – Top coordinate of the character in the bitmap font (in tiles)
- **img** (`Image`) – An image containing the new character bitmap.
- **x** (*int*) – Left pixel of the character in the image.
- **y** (*int*) – Top pixel of the character in the image.

`tcod.sys_register_SDL_renderer(callback: Callable[[Any], None])` → `None`  
Register a custom rendering function with libtcod.

---

**Note:** This callback will only be called by the SDL renderer.

---

The callack will receive a `CData void*` to an `SDL_Surface*` struct.

The callback is called on every call to `tcod.console_flush`.

**Parameters** `Callable[[CData], None]` (*callback*) – A function which takes a single argument.

`tcod.sys_check_for_event(mask: int, k: Optional[tcod.libtcodpy.Key], m: Optional[tcod.libtcodpy.Mouse])` → `int`  
Check for and return an event.

#### Parameters

- **mask** (*int*) – *Event Types* to wait for.
- **k** (`Optional[Key]`) – A `tcod.Key` instance which might be updated with an event. Can be `None`.
- **m** (`Optional[Mouse]`) – A `tcod.Mouse` instance which might be updated with an event. Can be `None`.

Deprecated since version 9.3: Use the `tcod.event.get` function to check for events.

`tcod.sys_wait_for_event` (*mask*: *int*, *k*: *Optional*[*tcod.libtcodpy.Key*], *m*: *Optional*[*tcod.libtcodpy.Mouse*], *flush*: *bool*) → *int*

Wait for an event then return.

If *flush* is *True* then the buffer will be cleared before waiting. Otherwise each available event will be returned in the order they're recieved.

#### Parameters

- **mask** (*int*) – *Event Types* to wait for.
- **k** (*Optional*[*Key*]) – A *tcod.Key* instance which might be updated with an event. Can be *None*.
- **m** (*Optional*[*Mouse*]) – A *tcod.Mouse* instance which might be updated with an event. Can be *None*.
- **flush** (*bool*) – Clear the event buffer before waiting.

Deprecated since version 9.3: Use the `tcod.event.wait` function to wait for events.

## 13.6 pathfinding

`tcod.dijkstra_compute` (*p*: *tcod.path.Dijkstra*, *ox*: *int*, *oy*: *int*) → *None*

`tcod.dijkstra_delete` (*p*: *tcod.path.Dijkstra*) → *None*

Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.dijkstra_get` (*p*: *tcod.path.Dijkstra*, *idx*: *int*) → *Tuple*[*int*, *int*]

`tcod.dijkstra_get_distance` (*p*: *tcod.path.Dijkstra*, *x*: *int*, *y*: *int*) → *int*

`tcod.dijkstra_is_empty` (*p*: *tcod.path.Dijkstra*) → *bool*

`tcod.dijkstra_new` (*m*: *tcod.map.Map*, *dcost*: *float* = 1.41) → *tcod.path.Dijkstra*

`tcod.dijkstra_new_using_function` (*w*: *int*, *h*: *int*, *func*: *Callable*[[*int*, *int*, *int*, *int*, *Any*], *float*], *userData*: *Any* = 0, *dcost*: *float* = 1.41) → *tcod.path.Dijkstra*

`tcod.dijkstra_path_set` (*p*: *tcod.path.Dijkstra*, *x*: *int*, *y*: *int*) → *bool*

`tcod.dijkstra_path_walk` (*p*: *tcod.path.Dijkstra*) → *Union*[*Tuple*[*int*, *int*], *Tuple*[*None*, *None*]]

`tcod.dijkstra_reverse` (*p*: *tcod.path.Dijkstra*) → *None*

`tcod.dijkstra_size` (*p*: *tcod.path.Dijkstra*) → *int*

`tcod.path_compute` (*p*: *tcod.path.AStar*, *ox*: *int*, *oy*: *int*, *dx*: *int*, *dy*: *int*) → *bool*

Find a path from (*ox*, *oy*) to (*dx*, *dy*). Return *True* if path is found.

#### Parameters

- **p** (*AStar*) – An *AStar* instance.
- **ox** (*int*) – Starting x position.
- **oy** (*int*) – Starting y position.
- **dx** (*int*) – Destination x position.
- **dy** (*int*) – Destination y position.

**Returns** *True* if a valid path was found. Otherwise *False*.

**Return type** `bool`

`tcod.path_delete` (*p*: `tcod.path.AStar`) → `None`

Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.path_get` (*p*: `tcod.path.AStar`, *idx*: `int`) → `Tuple[int, int]`

Get a point on a path.

**Parameters**

- **p** (`AStar`) – An `AStar` instance.
- **idx** (`int`) – Should be in range:  $0 \leq \text{idx} < \text{path\_size}$

`tcod.path_get_destination` (*p*: `tcod.path.AStar`) → `Tuple[int, int]`

Get the current destination position.

**Parameters** **p** (`AStar`) – An `AStar` instance.

**Returns** An (x, y) point.

**Return type** `Tuple[int, int]`

`tcod.path_get_origin` (*p*: `tcod.path.AStar`) → `Tuple[int, int]`

Get the current origin position.

This point moves when `path_walk` returns the next x,y step.

**Parameters** **p** (`AStar`) – An `AStar` instance.

**Returns** An (x, y) point.

**Return type** `Tuple[int, int]`

`tcod.path_is_empty` (*p*: `tcod.path.AStar`) → `bool`

Return True if a path is empty.

**Parameters** **p** (`AStar`) – An `AStar` instance.

**Returns** True if a path is empty. Otherwise False.

**Return type** `bool`

`tcod.path_new_using_function` (*w*: `int`, *h*: `int`, *func*: `Callable[[int, int, int, int, Any], float]`, *userData*: `Any = 0`, *dcost*: `float = 1.41`) → `tcod.path.AStar`

Return a new `AStar` using the given callable function.

**Parameters**

- **w** (`int`) – Clipping width.
- **h** (`int`) – Clipping height.
- **func** (`Callable[[int, int, int, int, Any], float]`) –
- **userData** (`Any`) –
- **dcost** (`float`) – A multiplier for the cost of diagonal movement. Can be set to 0 to disable diagonal movement.

**Returns** A new `AStar` instance.

**Return type** `AStar`

`tcod.path_new_using_map` (*m*: `tcod.map.Map`, *dcost*: `float = 1.41`) → `tcod.path.AStar`

Return a new `AStar` using the given `Map`.



**Parameters**

- **m** (`Map`) – A `Map` instance.
- **dcost** (`float`) – The path-finding cost of diagonal movement. Can be set to 0 to disable diagonal movement.

**Returns** A new `AStar` instance.

**Return type** `AStar`

`tcod.path.reverse` (*p*: `tcod.path.AStar`) → `None`  
Reverse the direction of a path.

This effectively swaps the origin and destination points.

**Parameters** **p** (`AStar`) – An `AStar` instance.

`tcod.path.size` (*p*: `tcod.path.AStar`) → `int`  
Return the current length of the computed path.

**Parameters** **p** (`AStar`) – An `AStar` instance.

**Returns** Length of the path.

**Return type** `int`

`tcod.path.walk` (*p*: `tcod.path.AStar`, *recompute*: `bool`) → `Union[Tuple[int, int], Tuple[None, None]]`  
Return the next (x, y) point in a path, or (None, None) if it's empty.

When `recompute` is `True` and a previously valid path reaches a point where it is now blocked, a new path will automatically be found.

**Parameters**

- **p** (`AStar`) – An `AStar` instance.
- **recompute** (`bool`) – Recompute the path automatically.

**Returns** A single (x, y) point, or (None, None)

**Return type** `Union[Tuple[int, int], Tuple[None, None]]`

## 13.7 heightmap

`tcod.heightmap.add` (*hm*: `numpy.ndarray`, *value*: `float`) → `None`  
Add value to all values on this heightmap.

**Parameters**

- **hm** (`numpy.ndarray`) – A `numpy.ndarray` formatted for heightmap functions.
- **value** (`float`) – A number to add to this heightmap.

Deprecated since version 2.0: Do `hm[: ] += value` instead.

`tcod.heightmap.add_fbm` (*hm*: `numpy.ndarray`, *noise*: `tcod.noise.Noise`, *mulx*: `float`, *muly*: `float`, *addx*: `float`, *addy*: `float`, *octaves*: `float`, *delta*: `float`, *scale*: `float`) → `None`  
Add FBM noise to the heightmap.

The noise coordinate for each map cell is  $((x + addx) * mulx / width, (y + addy) * muly / height)$ .

The value added to the heightmap is  $delta + noise * scale$ .

**Parameters**

- **hm** (*numpy.ndarray*) – A *numpy.ndarray* formatted for heightmap functions.
- **noise** (*Noise*) – A *Noise* instance.
- **mulx** (*float*) – Scaling of each x coordinate.
- **muly** (*float*) – Scaling of each y coordinate.
- **addx** (*float*) – Translation of each x coordinate.
- **addy** (*float*) – Translation of each y coordinate.
- **octaves** (*float*) – Number of octaves in the FBM sum.
- **delta** (*float*) – The value added to all heightmap cells.
- **scale** (*float*) – The noise value is scaled with this parameter.

Deprecated since version 8.1: An equivalent array of noise samples can be taken using a method such as *Noise.sample\_ogrid*.

`tcod.heightmap_add_hill` (*hm: numpy.ndarray, x: float, y: float, radius: float, height: float*) → None  
Add a hill (a half spheroid) at given position.

If `height == radius` or `-radius`, the hill is a half-sphere.

#### Parameters

- **hm** (*numpy.ndarray*) – A *numpy.ndarray* formatted for heightmap functions.
- **x** (*float*) – The x position at the center of the new hill.
- **y** (*float*) – The y position at the center of the new hill.
- **radius** (*float*) – The size of the new hill.
- **height** (*float*) – The height or depth of the new hill.

`tcod.heightmap_add_hm` (*hm1: numpy.ndarray, hm2: numpy.ndarray, hm3: numpy.ndarray*) → None  
Add two heightmaps together and stores the result in *hm3*.

#### Parameters

- **hm1** (*numpy.ndarray*) – The first heightmap.
- **hm2** (*numpy.ndarray*) – The second heightmap to add to the first.
- **hm3** (*numpy.ndarray*) – A destination heightmap to store the result.

Deprecated since version 2.0: Do `hm3[:] = hm1[:] + hm2[:]` instead.

`tcod.heightmap_add_voronoi` (*hm: numpy.ndarray, nbPoints: Any, nbCoef: int, coef: Sequence[float], rnd: Optional[tcod.random.Random] = None*) → None  
Add values from a Voronoi diagram to the heightmap.

#### Parameters

- **hm** (*numpy.ndarray*) – A *numpy.ndarray* formatted for heightmap functions.
- **nbPoints** (*Any*) – Number of Voronoi sites.
- **nbCoef** (*int*) – The diagram value is calculated from the *nbCoef* closest sites.
- **coef** (*Sequence[float]*) – The distance to each site is scaled by the corresponding *coef*. Closest site : *coef*[0], second closest site : *coef*[1], ...
- **rnd** (*Optional[Random]*) – A *Random* instance, or *None*.

`tcod.heightmap_clamp` (*hm: numpy.ndarray, mi: float, ma: float*) → None  
Clamp all values on this heightmap between *mi* and *ma*

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **mi** (*float*) – The lower bound to clamp to.
- **ma** (*float*) – The upper bound to clamp to.

Deprecated since version 2.0: Do `hm.clip(mi, ma)` instead.

`tcod.heightmap_clear` (*hm: numpy.ndarray*) → None

Add value to all values on this heightmap.

**Parameters** **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.

Deprecated since version 2.0: Do `hm.array[:] = 0` instead.

`tcod.heightmap_copy` (*hm1: numpy.ndarray, hm2: numpy.ndarray*) → None

Copy the heightmap `hm1` to `hm2`.

**Parameters**

- **hm1** (*numpy.ndarray*) – The source heightmap.
- **hm2** (*numpy.ndarray*) – The destination heightmap.

Deprecated since version 2.0: Do `hm2[:] = hm1[:]` instead.

`tcod.heightmap_count_cells` (*hm: numpy.ndarray, mi: float, ma: float*) → int

Return the number of map cells which value is between `mi` and `ma`.

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **mi** (*float*) – The lower bound.
- **ma** (*float*) – The upper bound.

**Returns** The count of values which fall between `mi` and `ma`.

**Return type** int

Deprecated since version 8.1: Can be replaced by an equivalent NumPy function such as: `numpy.count_nonzero((mi <= hm) & (hm < ma))`

`tcod.heightmap_delete` (*hm: Any*) → None

Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

Deprecated since version 2.0: libtcod-ffi deletes heightmaps automatically.

`tcod.heightmap_dig_bezier` (*hm: numpy.ndarray, px: Tuple[int, int, int, int], py: Tuple[int, int, int, int], startRadius: float, startDepth: float, endRadius: float, endDepth: float*) → None

Carve a path along a cubic Bezier curve.

Both radius and depth can vary linearly along the path.

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **px** (*Sequence[int]*) – The 4 *x* coordinates of the Bezier curve.
- **py** (*Sequence[int]*) – The 4 *y* coordinates of the Bezier curve.
- **startRadius** (*float*) – The starting radius size.

- **startDepth** (*float*) – The starting depth.
- **endRadius** (*float*) – The ending radius size.
- **endDepth** (*float*) – The ending depth.

`tcod.heightmap_dig_hill` (*hm: numpy.ndarray, x: float, y: float, radius: float, height: float*) → None  
This function takes the highest value (if height > 0) or the lowest (if height < 0) between the map and the hill.

It's main goal is to carve things in maps (like rivers) by digging hills along a curve.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **x** (*float*) – The x position at the center of the new carving.
- **y** (*float*) – The y position at the center of the new carving.
- **radius** (*float*) – The size of the carving.
- **height** (*float*) – The height or depth of the hill to dig out.

`tcod.heightmap_get_interpolated_value` (*hm: numpy.ndarray, x: float, y: float*) → float  
Return the interpolated height at non integer coordinates.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **x** (*float*) – A floating point x coordinate.
- **y** (*float*) – A floating point y coordinate.

**Returns** The value at x, y.

**Return type** float

`tcod.heightmap_get_minmax` (*hm: numpy.ndarray*) → Tuple[float, float]  
Return the min and max values of this heightmap.

**Parameters** **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.

**Returns** The (min, max) values.

**Return type** Tuple[float, float]

Deprecated since version 2.0: Use `hm.min()` or `hm.max()` instead.

`tcod.heightmap_get_normal` (*hm: numpy.ndarray, x: float, y: float, waterLevel: float*) → Tuple[float, float, float]  
Return the map normal at given coordinates.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **x** (*float*) – The x coordinate.
- **y** (*float*) – The y coordinate.
- **waterLevel** (*float*) – The heightmap is considered flat below this value.

**Returns** An (x, y, z) vector normal.

**Return type** Tuple[float, float, float]

`tcod.heightmap_get_slope` (*hm: numpy.ndarray, x: int, y: int*) → float  
Return the slope between 0 and (pi / 2) at given coordinates.

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **x** (*int*) – The x coordinate.
- **y** (*int*) – The y coordinate.

**Returns** The steepness at `x`, `y`. From 0 to  $(\pi / 2)$

**Return type** `float`

`tcod.heightmap_get_value` (*hm: numpy.ndarray, x: int, y: int*) → `float`  
Return the value at `x`, `y` in a heightmap.

Deprecated since version 2.0: Access `hm` as a NumPy array instead.

`tcod.heightmap_has_land_on_border` (*hm: numpy.ndarray, waterlevel: float*) → `bool`  
Returns True if the map edges are below `waterlevel`, otherwise False.

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **waterLevel** (*float*) – The water level to use.

**Returns** True if the map edges are below `waterlevel`, otherwise False.

**Return type** `bool`

`tcod.heightmap_kernel_transform` (*hm: numpy.ndarray, kernelSize: int, dx: Sequence[int], dy: Sequence[int], weight: Sequence[float], minLevel: float, maxLevel: float*) → `None`

Apply a generic transformation on the map, so that each resulting cell value is the weighted sum of several neighbour cells.

This can be used to smooth/sharpen the map.

**Parameters**

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **kernelSize** (*int*) –  
**Should be set to the length of the parameters::** `dx`, `dy`, and `weight`.
- **dx** (*Sequence[int]*) – A sequence of x coordinates.
- **dy** (*Sequence[int]*) – A sequence of y coordinates.
- **weight** (*Sequence[float]*) – A sequence of `kernelSize` cells weight. The value of each neighbour cell is scaled by its corresponding weight
- **minLevel** (*float*) – No transformation will apply to cells below this value.
- **maxLevel** (*float*) – No transformation will apply to cells above this value.

See examples below for a simple horizontal smoothing kernel : replace `value(x,y)` with `0.33*value(x-1,y) + 0.33*value(x,y) + 0.33*value(x+1,y)`. To do this, you need a kernel of size 3 (the sum involves 3 surrounding cells). The `dx,dy` array will contain:

- `dx=-1, dy=0` for cell `(x-1, y)`
- `dx=1, dy=0` for cell `(x+1, y)`
- `dx=0, dy=0` for cell `(x, y)`
- The weight array will contain 0.33 for each cell.

## Example

```
>>> import numpy as np
>>> heightmap = np.zeros((3, 3), dtype=np.float32)
>>> heightmap[:,1] = 1
>>> dx = [-1, 1, 0]
>>> dy = [0, 0, 0]
>>> weight = [0.33, 0.33, 0.33]
>>> tcod.heightmap_kernel_transform(heightmap, 3, dx, dy, weight,
...                               0.0, 1.0)
```

`tcod.heightmap_lerp_hm` (*hm1*: *numpy.ndarray*, *hm2*: *numpy.ndarray*, *hm3*: *numpy.ndarray*, *coef*: *float*) → *None*

Perform linear interpolation between two heightmaps storing the result in *hm3*.

This is the same as doing `hm3[:] = hm1[:] + (hm2[:] - hm1[:]) * coef`

### Parameters

- **hm1** (*numpy.ndarray*) – The first heightmap.
- **hm2** (*numpy.ndarray*) – The second heightmap to add to the first.
- **hm3** (*numpy.ndarray*) – A destination heightmap to store the result.
- **coef** (*float*) – The linear interpolation coefficient.

`tcod.heightmap_multiply_hm` (*hm1*: *numpy.ndarray*, *hm2*: *numpy.ndarray*, *hm3*: *numpy.ndarray*) → *None*

Multiplies two heightmap's together and stores the result in *hm3*.

### Parameters

- **hm1** (*numpy.ndarray*) – The first heightmap.
- **hm2** (*numpy.ndarray*) – The second heightmap to multiply with the first.
- **hm3** (*numpy.ndarray*) – A destination heightmap to store the result.

Deprecated since version 2.0: Do `hm3[:] = hm1[:] * hm2[:]` instead. Alternatively you can do `HeightMap(hm1.array[:] * hm2.array[:])`.

`tcod.heightmap_new` (*w*: *int*, *h*: *int*, *order*: *str* = 'C') → *numpy.ndarray*

Return a new *numpy.ndarray* formatted for use with heightmap functions.

*w* and *h* are the width and height of the array.

*order* is given to the new NumPy array, it can be 'C' or 'F'.

You can pass a NumPy array to any heightmap function as long as all the following are true:: \* The array is 2 dimensional. \* The array has the C\_CONTIGUOUS or F\_CONTIGUOUS flag. \* The array's dtype is `dtype.float32`.

The returned NumPy array will fit all these conditions.

Changed in version 8.1: Added the *order* parameter.

`tcod.heightmap_normalize` (*hm*: *numpy.ndarray*, *mi*: *float* = 0.0, *ma*: *float* = 1.0) → *None*

Normalize heightmap values between *mi* and *ma*.

### Parameters

- **mi** (*float*) – The lowest value after normalization.
- **ma** (*float*) – The highest value after normalization.

`tcod.heightmap_rain_erosion` (*hm: numpy.ndarray, nbDrops: int, erosionCoef: float, sedimentationCoef: float, rnd: Optional[tcod.random.Random] = None*) → None  
 Simulate the effect of rain drops on the terrain, resulting in erosion.

`nbDrops` should be at least `hm.size`.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **nbDrops** (*int*) – Number of rain drops to simulate.
- **erosionCoef** (*float*) – Amount of ground eroded on the drop’s path.
- **sedimentationCoef** (*float*) – Amount of ground deposited when the drops stops to flow.
- **rnd** (*Optional[Random]*) – A `tcod.Random` instance, or None.

`tcod.heightmap_scale` (*hm: numpy.ndarray, value: float*) → None  
 Multiply all items on this heightmap by value.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **value** (*float*) – A number to scale this heightmap by.

Deprecated since version 2.0: Do `hm[:] *= value` instead.

`tcod.heightmap_scale_fbm` (*hm: numpy.ndarray, noise: tcod.noise.Noise, mulx: float, muly: float, addx: float, addy: float, octaves: float, delta: float, scale: float*) → None  
 Multiply the heighmap values with FBM noise.

#### Parameters

- **hm** (*numpy.ndarray*) – A `numpy.ndarray` formatted for heightmap functions.
- **noise** (*Noise*) – A `Noise` instance.
- **mulx** (*float*) – Scaling of each x coordinate.
- **muly** (*float*) – Scaling of each y coordinate.
- **addx** (*float*) – Translation of each x coordinate.
- **addy** (*float*) – Translation of each y coordinate.
- **octaves** (*float*) – Number of octaves in the FBM sum.
- **delta** (*float*) – The value added to all heightmap cells.
- **scale** (*float*) – The noise value is scaled with this parameter.

Deprecated since version 8.1: An equivalent array of noise samples can be taken using a method such as `Noise.sample_ogrid`.

`tcod.heightmap_set_value` (*hm: numpy.ndarray, x: int, y: int, value: float*) → None  
 Set the value of a point on a heightmap.

Deprecated since version 2.0: `hm` is a NumPy array, so values should be assigned to it directly.

## 13.8 image

`tcod.image_load` (*filename: str*) → `tcod.image.Image`  
 Load an image file into an `Image` instance and return it.

**Parameters** `filename` (*AnyStr*) – Path to a .bmp or .png image file.

`tcod.image_from_console` (*console: tcod.console.Console*) → `tcod.image.Image`  
Return an Image with a Consoles pixel data.

This effectively takes a screen-shot of the Console.

**Parameters** `console` (*Console*) – Any Console instance.

`tcod.image_blit` (*image: tcod.image.Image, console: tcod.console.Console, x: float, y: float, bkgnd\_flag: int, scalex: float, scaley: float, angle: float*) → `None`

`tcod.image_blit_2x` (*image: tcod.image.Image, console: tcod.console.Console, dx: int, dy: int, sx: int = 0, sy: int = 0, w: int = -1, h: int = -1*) → `None`

`tcod.image_blit_rect` (*image: tcod.image.Image, console: tcod.console.Console, x: int, y: int, w: int, h: int, bkgnd\_flag: int*) → `None`

`tcod.image_clear` (*image: tcod.image.Image, col: Tuple[int, int, int]*) → `None`

`tcod.image_delete` (*image: tcod.image.Image*) → `None`  
Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.image_get_alpha` (*image: tcod.image.Image, x: int, y: int*) → `int`

`tcod.image_get_mipmap_pixel` (*image: tcod.image.Image, x0: float, y0: float, x1: float, y1: float*) → `Tuple[int, int, int]`

`tcod.image_get_pixel` (*image: tcod.image.Image, x: int, y: int*) → `Tuple[int, int, int]`

`tcod.image_get_size` (*image: tcod.image.Image*) → `Tuple[int, int]`

`tcod.image_hflip` (*image: tcod.image.Image*) → `None`

`tcod.image_invert` (*image: tcod.image.Image*) → `None`

`tcod.image_is_pixel_transparent` (*image: tcod.image.Image, x: int, y: int*) → `bool`

`tcod.image_new` (*width: int, height: int*) → `tcod.image.Image`

`tcod.image_put_pixel` (*image: tcod.image.Image, x: int, y: int, col: Tuple[int, int, int]*) → `None`

`tcod.image_refresh_console` (*image: tcod.image.Image, console: tcod.console.Console*) → `None`

`tcod.image_rotate90` (*image: tcod.image.Image, num: int = 1*) → `None`

`tcod.image_save` (*image: tcod.image.Image, filename: str*) → `None`

`tcod.image_scale` (*image: tcod.image.Image, neww: int, newh: int*) → `None`

`tcod.image_set_key_color` (*image: tcod.image.Image, col: Tuple[int, int, int]*) → `None`

`tcod.image_vflip` (*image: tcod.image.Image*) → `None`

## 13.9 line

`tcod.line_init` (*xo: int, yo: int, xd: int, yd: int*) → `None`  
Initilize a line whose points will be returned by `line_step`.

This function does not return anything on its own.

Does not include the origin point.

**Parameters**



- `xo(int)` – X starting point.
- `yo(int)` – Y starting point.
- `xd(int)` – X destination point.
- `yd(int)` – Y destination point.

Deprecated since version 2.0: Use `line_iter` instead.

`tcod.line_step()` → Union[Tuple[int, int], Tuple[None, None]]

After calling `line_init` returns (x, y) points of the line.

Once all points are exhausted this function will return (None, None)

**Returns** The next (x, y) point of the line setup by `line_init`, or (None, None) if there are no more points.

**Return type** Union[Tuple[int, int], Tuple[None, None]]

Deprecated since version 2.0: Use `line_iter` instead.

`tcod.line(xo: int, yo: int, xd: int, yd: int, py_callback: Callable[[int, int], bool])` → bool

Iterate over a line using a callback function.

Your callback function will take x and y parameters and return True to continue iteration or False to stop iteration and return.

This function includes both the start and end points.

#### Parameters

- `xo(int)` – X starting point.
- `yo(int)` – Y starting point.
- `xd(int)` – X destination point.
- `yd(int)` – Y destination point.
- `py_callback(Callable[[int, int], bool])` – A callback which takes x and y parameters and returns bool.

#### Returns

**False if the callback cancels the line iteration by** returning False or None, otherwise True.

**Return type** bool

Deprecated since version 2.0: Use `line_iter` instead.

`tcod.line_iter(xo: int, yo: int, xd: int, yd: int)` → Iterator[Tuple[int, int]]

returns an Iterable

This Iterable does not include the origin point.

#### Parameters

- `xo(int)` – X starting point.
- `yo(int)` – Y starting point.
- `xd(int)` – X destination point.
- `yd(int)` – Y destination point.

**Returns** An Iterable of (x,y) points.

**Return type** Iterable[Tuple[int,int]]

`tcod.line_where` (*x1*: int, *y1*: int, *x2*: int, *y2*: int, *inclusive*: bool = True) → Tuple[numpy.ndarray, numpy.ndarray]

Return a NumPy index array following a Bresenham line.

If *inclusive* is true then the start point is included in the result.

### Example

```
>>> where = tcod.line_where(1, 0, 3, 4)
>>> where
(array([1, 1, 2, 2, 3]...), array([0, 1, 2, 3, 4]...))
>>> array = np.zeros((5, 5), dtype=np.int32)
>>> array[where] = np.arange(len(where[0])) + 1
>>> array
array([[0, 0, 0, 0, 0],
       [1, 2, 0, 0, 0],
       [0, 0, 3, 4, 0],
       [0, 0, 0, 0, 5],
       [0, 0, 0, 0, 0]]...)
```

New in version 4.6.

## 13.10 map

`tcod.map_clear` (*m*: `tcod.map.Map`, *transparent*: bool = False, *walkable*: bool = False) → None

Change all map cells to a specific value.

Deprecated since version 4.5: Use `tcod.map.Map.transparent` and `tcod.map.Map.walkable` arrays to set these properties.

`tcod.map_compute_fov` (*m*: `tcod.map.Map`, *x*: int, *y*: int, *radius*: int = 0, *light\_walls*: bool = True, *algo*: int = 12) → None

Compute the field-of-view for a map instance.

Deprecated since version 4.5: Use `tcod.map.Map.compute_fov` instead.

`tcod.map_copy` (*source*: `tcod.map.Map`, *dest*: `tcod.map.Map`) → None

Copy map data from *source* to *dest*.

Deprecated since version 4.5: Use Python's copy module, or see `tcod.map.Map` and assign between array attributes manually.

`tcod.map_delete` (*m*: `tcod.map.Map`) → None

Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.map_get_height` (*map*: `tcod.map.Map`) → int

Return the height of a map.

Deprecated since version 4.5: Check the `tcod.map.Map.height` attribute instead.

`tcod.map_get_width` (*map*: `tcod.map.Map`) → int

Return the width of a map.

Deprecated since version 4.5: Check the `tcod.map.Map.width` attribute instead.

`tcod.map_is_in_fov` (*m: tcod.map.Map, x: int, y: int*) → bool  
Return True if the cell at x,y is lit by the last field-of-view algorithm.

---

**Note:** This function is slow.

---

Deprecated since version 4.5: Use `tcod.map.Map.fov` to check this property.

`tcod.map_is_transparent` (*m: tcod.map.Map, x: int, y: int*) → bool

---

**Note:** This function is slow.

---

Deprecated since version 4.5: Use `tcod.map.Map.transparent` to check this property.

`tcod.map_is_walkable` (*m: tcod.map.Map, x: int, y: int*) → bool

---

**Note:** This function is slow.

---

Deprecated since version 4.5: Use `tcod.map.Map.walkable` to check this property.

`tcod.map_new` (*w: int, h: int*) → `tcod.map.Map`  
Return a `tcod.map.Map` with a width and height.

Deprecated since version 4.5: Use the `tcod.map` module for working with field-of-view, or `tcod.path` for working with path-finding.

`tcod.map_set_properties` (*m: tcod.map.Map, x: int, y: int, isTrans: bool, isWalk: bool*) → None  
Set the properties of a single cell.

---

**Note:** This function is slow.

---

Deprecated since version 4.5: Use `tcod.map.Map.transparent` and `tcod.map.Map.walkable` arrays to set these properties.

## 13.11 mouse

`tcod.mouse_get_status` () → `tcod.libtcodpy.Mouse`

`tcod.mouse_is_cursor_visible` () → bool  
Return True if the mouse cursor is visible.

`tcod.mouse_move` (*x: int, y: int*) → None

`tcod.mouse_show_cursor` (*visible: bool*) → None  
Change the visibility of the mouse cursor.

## 13.12 namegen

`tcod.namegen_destroy` () → None

`tcod.namegen_generate` (*name: str*) → str

`tcod.namegen_generate_custom` (*name: str, rule: str*) → str

`tcod.namegen_get_sets` () → List[str]

`tcod.namegen_parse` (*filename: str, random: Optional[tcod.random.Random] = None*) → None

## 13.13 noise

`tcod.noise_delete` (*n: tcod.noise.Noise*) → None

Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.noise_get` (*n: tcod.noise.Noise, f: Sequence[float], typ: int = 0*) → float

Return the noise value sampled from the `f` coordinate.

`f` should be a tuple or list with a length matching `Noise.dimensions`. If `f` is shorter than `Noise.dimensions` the missing coordinates will be filled with zeros.

### Parameters

- `n` (`Noise`) – A `Noise` instance.
- `f` (`Sequence[float]`) – The point to sample the noise from.
- `typ` (`int`) – The noise algorithm to use.

**Returns** The sampled noise value.

**Return type** float

`tcod.noise_get_fbm` (*n: tcod.noise.Noise, f: Sequence[float], oc: float, typ: int = 0*) → float

Return the fractal Brownian motion sampled from the `f` coordinate.

### Parameters

- `n` (`Noise`) – A `Noise` instance.
- `f` (`Sequence[float]`) – The point to sample the noise from.
- `typ` (`int`) – The noise algorithm to use.
- `octaves` (`float`) – The level of level. Should be more than 1.

**Returns** The sampled noise value.

**Return type** float

`tcod.noise_get_turbulence` (*n: tcod.noise.Noise, f: Sequence[float], oc: float, typ: int = 0*) → float

Return the turbulence noise sampled from the `f` coordinate.

### Parameters

- `n` (`Noise`) – A `Noise` instance.
- `f` (`Sequence[float]`) – The point to sample the noise from.
- `typ` (`int`) – The noise algorithm to use.
- `octaves` (`float`) – The level of level. Should be more than 1.

**Returns** The sampled noise value.

**Return type** float

`tcod.noise_new` (*dim*: int, *h*: float = 0.5, *l*: float = 2.0, *random*: Optional[`tcod.random.Random`] = None) → `tcod.noise.Noise`  
 Return a new Noise instance.

**Parameters**

- **dim** (*int*) – Number of dimensions. From 1 to 4.
- **h** (*float*) – The hurst exponent. Should be in the 0.0-1.0 range.
- **l** (*float*) – The noise lacunarity.
- **random** (*Optional*[`Random`]) – A `Random` instance, or `None`.

**Returns** The new Noise instance.

**Return type** `Noise`

`tcod.noise_set_type` (*n*: `tcod.noise.Noise`, *typ*: int) → None  
 Set a Noise objects default noise algorithm.

**Parameters** **typ** (*int*) – Any `NOISE_*` constant.

## 13.14 parser

`tcod.parser_delete` (*parser*: Any) → None  
 Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.parser_get_bool_property` (*parser*: Any, *name*: str) → bool

`tcod.parser_get_char_property` (*parser*: Any, *name*: str) → str

`tcod.parser_get_color_property` (*parser*: Any, *name*: str) → `tcod.color.Color`

`tcod.parser_get_dice_property` (*parser*: Any, *name*: str) → `tcod.libtcodpy.Dice`

`tcod.parser_get_float_property` (*parser*: Any, *name*: str) → float

`tcod.parser_get_int_property` (*parser*: Any, *name*: str) → int

`tcod.parser_get_list_property` (*parser*: Any, *name*: str, *type*: Any) → Any

`tcod.parser_get_string_property` (*parser*: Any, *name*: str) → str

`tcod.parser_new` () → Any

`tcod.parser_new_struct` (*parser*: Any, *name*: str) → Any

`tcod.parser_run` (*parser*: Any, *filename*: str, *listener*: Any = None) → None

## 13.15 random

`tcod.random_delete` (*rnd*: `tcod.random.Random`) → None  
 Does nothing. libtcod objects are managed by Python's garbage collector.

This function exists for backwards compatibility with libtcodpy.

`tcod.random_get_double` (*rnd: Optional[tcod.random.Random], mi: float, ma: float*) → float  
Return a random float in the range:  $mi \leq n \leq ma$ .

Deprecated since version 2.0: Use `random_get_float` instead. Both funtions return a double precision float.

`tcod.random_get_double_mean` (*rnd: Optional[tcod.random.Random], mi: float, ma: float, mean: float*) → float  
Return a random weighted float in the range:  $mi \leq n \leq ma$ .

Deprecated since version 2.0: Use `random_get_float_mean` instead. Both funtions return a double precision float.

`tcod.random_get_float` (*rnd: Optional[tcod.random.Random], mi: float, ma: float*) → float  
Return a random float in the range:  $mi \leq n \leq ma$ .

The result is affected by calls to `random_set_distribution`.

#### Parameters

- **rnd** (*Optional[Random]*) – A Random instance, or None to use the default.
- **low** (*float*) – The lower bound of the random range, inclusive.
- **high** (*float*) – The upper bound of the random range, inclusive.

#### Returns

A random double precision float in the range  $mi \leq n \leq ma$ .

#### Return type

 float

`tcod.random_get_float_mean` (*rnd: Optional[tcod.random.Random], mi: float, ma: float, mean: float*) → float  
Return a random weighted float in the range:  $mi \leq n \leq ma$ .

The result is affacted by calls to `random_set_distribution`.

#### Parameters

- **rnd** (*Optional[Random]*) – A Random instance, or None to use the default.
- **low** (*float*) – The lower bound of the random range, inclusive.
- **high** (*float*) – The upper bound of the random range, inclusive.
- **mean** (*float*) – The mean return value.

#### Returns

A random weighted double precision float in the range  $mi \leq n \leq ma$ .

#### Return type

 float

`tcod.random_get_instance` () → `tcod.random.Random`  
Return the default Random instance.

**Returns** A Random instance using the default random number generator.

#### Return type

*Random*

`tcod.random_get_int` (*rnd: Optional[tcod.random.Random], mi: int, ma: int*) → int  
Return a random integer in the range:  $mi \leq n \leq ma$ .

The result is affected by calls to `random_set_distribution`.

#### Parameters

- **rnd** (*Optional[Random]*) – A Random instance, or None to use the default.

- **low** (*int*) – The lower bound of the random range, inclusive.
- **high** (*int*) – The upper bound of the random range, inclusive.

**Returns** A random integer in the range  $m_i \leq n \leq m_a$ .

**Return type** *int*

`tcod.random_get_int_mean` (*rnd: Optional[tcod.random.Random]*, *mi: int*, *ma: int*, *mean: int*) → *int*  
Return a random weighted integer in the range:  $m_i \leq n \leq m_a$ .

The result is affected by calls to *random\_set\_distribution*.

**Parameters**

- **rnd** (*Optional[Random]*) – A Random instance, or None to use the default.
- **low** (*int*) – The lower bound of the random range, inclusive.
- **high** (*int*) – The upper bound of the random range, inclusive.
- **mean** (*int*) – The mean return value.

**Returns** A random weighted integer in the range  $m_i \leq n \leq m_a$ .

**Return type** *int*

`tcod.random_new` (*algo: int = 1*) → *tcod.random.Random*  
Return a new Random instance. Using *algo*.

**Parameters** **algo** (*int*) – The random number algorithm to use.

**Returns** A new Random instance using the given algorithm.

**Return type** *Random*

`tcod.random_new_from_seed` (*seed: Hashable*, *algo: int = 1*) → *tcod.random.Random*  
Return a new Random instance. Using the given *seed* and *algo*.

**Parameters**

- **seed** (*Hashable*) – The RNG seed. Should be a 32-bit integer, but any hashable object is accepted.
- **algo** (*int*) – The random number algorithm to use.

**Returns** A new Random instance using the given algorithm.

**Return type** *Random*

`tcod.random_restore` (*rnd: Optional[tcod.random.Random]*, *backup: tcod.random.Random*) → *None*  
Restore a random number generator from a backed up copy.

**Parameters**

- **rnd** (*Optional[Random]*) – A Random instance, or None to use the default.
- **backup** (*Random*) – The Random instance which was used as a backup.

Deprecated since version 8.4: You can use the standard library copy and pickle modules to save a random state.

`tcod.random_save` (*rnd: Optional[tcod.random.Random]*) → *tcod.random.Random*  
Return a copy of a random number generator.

Deprecated since version 8.4: You can use the standard library copy and pickle modules to save a random state.

`tcod.random_set_distribution` (*rnd: Optional[tcod.random.Random]*, *dist: int*) → *None*  
Change the distribution mode of a random number generator.

**Parameters**

- **rnd** (*Optional [Random]*) – A `Random` instance, or `None` to use the default.
- **dist** (*int*) – The distribution mode to use. Should be `DISTRIBUTION_*`.

## 13.16 struct

`tcod.struct_add_flag` (*struct, name*)

`tcod.struct_add_list_property` (*struct, name, typ, mandatory*)

`tcod.struct_add_property` (*struct, name, typ, mandatory*)

`tcod.struct_add_structure` (*struct, sub\_struct*)

`tcod.struct_add_value_list` (*struct, name, value\_list, mandatory*)

`tcod.struct_get_name` (*struct*)

`tcod.struct_get_type` (*struct, name*)

`tcod.struct_is_mandatory` (*struct, name*)

## 13.17 other

**class** `tcod.ConsoleBuffer` (*width: int, height: int, back\_r: int = 0, back\_g: int = 0, back\_b: int = 0, fore\_r: int = 0, fore\_g: int = 0, fore\_b: int = 0, char: str = ''*)

Simple console that allows direct (fast) access to cells. simplifies use of the “fill” functions.

Deprecated since version 6.0: Console array attributes perform better than this class.

**Parameters**

- **width** (*int*) – Width of the new `ConsoleBuffer`.
- **height** (*int*) – Height of the new `ConsoleBuffer`.
- **back\_r** (*int*) – Red background color, from 0 to 255.
- **back\_g** (*int*) – Green background color, from 0 to 255.
- **back\_b** (*int*) – Blue background color, from 0 to 255.
- **fore\_r** (*int*) – Red foreground color, from 0 to 255.
- **fore\_g** (*int*) – Green foreground color, from 0 to 255.
- **fore\_b** (*int*) – Blue foreground color, from 0 to 255.
- **char** (*AnyStr*) – A single character `str` or `bytes` object.

**blit** (*dest: tcod.console.Console, fill\_fore: bool = True, fill\_back: bool = True*) → `None`

Use `libtcod`’s “fill” functions to write the buffer to a console.

**Parameters**

- **dest** (`Console`) – Console object to modify.
- **fill\_fore** (`bool`) – If `True`, fill the foreground color and characters.
- **fill\_back** (`bool`) – If `True`, fill the background color.



**clear** (*back\_r: int = 0, back\_g: int = 0, back\_b: int = 0, fore\_r: int = 0, fore\_g: int = 0, fore\_b: int = 0, char: str = ' ')* → None  
 Clears the console. Values to fill it with are optional, defaults to black with no characters.

#### Parameters

- **back\_r** (*int*) – Red background color, from 0 to 255.
- **back\_g** (*int*) – Green background color, from 0 to 255.
- **back\_b** (*int*) – Blue background color, from 0 to 255.
- **fore\_r** (*int*) – Red foreground color, from 0 to 255.
- **fore\_g** (*int*) – Green foreground color, from 0 to 255.
- **fore\_b** (*int*) – Blue foreground color, from 0 to 255.
- **char** (*AnyStr*) – A single character str or bytes object.

**copy** () → `tcod.libtcodpy.ConsoleBuffer`  
 Returns a copy of this ConsoleBuffer.

**Returns** A new ConsoleBuffer copy.

**Return type** *ConsoleBuffer*

**set** (*x: int, y: int, back\_r: int, back\_g: int, back\_b: int, fore\_r: int, fore\_g: int, fore\_b: int, char: str*) → None  
 Set the background color, foreground color and character of one cell.

#### Parameters

- **x** (*int*) – X position to change.
- **y** (*int*) – Y position to change.
- **back\_r** (*int*) – Red background color, from 0 to 255.
- **back\_g** (*int*) – Green background color, from 0 to 255.
- **back\_b** (*int*) – Blue background color, from 0 to 255.
- **fore\_r** (*int*) – Red foreground color, from 0 to 255.
- **fore\_g** (*int*) – Green foreground color, from 0 to 255.
- **fore\_b** (*int*) – Blue foreground color, from 0 to 255.
- **char** (*AnyStr*) – A single character str or bytes object.

**set\_back** (*x: int, y: int, r: int, g: int, b: int*) → None  
 Set the background color of one cell.

#### Parameters

- **x** (*int*) – X position to change.
- **y** (*int*) – Y position to change.
- **r** (*int*) – Red background color, from 0 to 255.
- **g** (*int*) – Green background color, from 0 to 255.
- **b** (*int*) – Blue background color, from 0 to 255.

**set\_fore** (*x: int, y: int, r: int, g: int, b: int, char: str*) → None  
 Set the character and foreground color of one cell.

#### Parameters

- **x** (*int*) – X position to change.
- **y** (*int*) – Y position to change.
- **r** (*int*) – Red foreground color, from 0 to 255.
- **g** (*int*) – Green foreground color, from 0 to 255.
- **b** (*int*) – Blue foreground color, from 0 to 255.
- **char** (*AnyStr*) – A single character str or bytes object.

**class** `tcod.Dice` (*nb\_dices=0, nb\_faces=0, multiplier=0, addsub=0*)

**Parameters**

- **nb\_dices** (*int*) – Number of dice.
- **nb\_faces** (*int*) – Number of sides on a die.
- **multiplier** (*float*) – Multiplier.
- **addsub** (*float*) – Addition.

Deprecated since version 2.0: You should make your own dice functions instead of using this class which is tied to a CData object.

Deprecated since version 8.4: This module has been deprecated.

## 14.1 Getting Started

Once the library is imported you can load the font you want to use with `tdl.set_font`. This is optional and when skipped will use a decent default font.

After that you call `tdl.init` to set the size of the window and get the root console in return. This console is the canvas to what will appear on the screen.

## 14.2 Indexing Consoles

For most methods taking a position you can use Python-style negative indexes to refer to the opposite side of a console with `(-1, -1)` starting at the bottom right. You can also check if a point is part of a console using containment logic i.e. `((x, y) in console)`.

You may also iterate over a console using a for statement. This returns every x,y coordinate available to draw on but it will be extremely slow to actually operate on every coordinate individually. Try to minimize draws by using an offscreen `Console`, only drawing what needs to be updated, and using `Console.blit`.

## 14.3 Drawing and Colors

Once you have the root console from `tdl.init` you can start drawing on it using a method such as `Console.draw_char`. When using this method you can have the char parameter be an integer or a single character string.

The `fg` and `bg` parameters expect a variety of types. The parameters default to Ellipsis which will tell the function to use the colors previously set by the `Console.set_colors` method. The colors set by

:any 'Console.set\_colors' are per each *Console/Window* and default to white on black. You can use a 3-item list/tuple of [red, green, blue] with integers in the 0-255 range with [0, 0, 0] being black and [255, 255, 255] being white. You can even use a single integer of 0xRRGGBB if you like.

Using None in the place of any of the three parameters (char, fg, bg) will tell the function to not overwrite that color or character.

After the drawing functions are called a call to `tdl.flush` will update the screen.

## 14.4 tdl API

`tdl.set_font` (*path*, *columns=None*, *rows=None*, *columnFirst=False*, *greyscale=False*, *altLayout=False*)

Changes the font to be used for this session. This should be called before `tdl.init`

If the font specifies its size in its filename (i.e. font\_NxN.png) then this function can auto-detect the tileset formatting and the parameters columns and rows can be left None.

While it's possible you can change the font mid program it can sometimes break in rare circumstances. So use caution when doing this.

### Parameters

- **path** (*Text*) – A file path to a *.bmp* or *.png* file.
- **columns** (*int*) – Number of columns in the tileset.  
Can be left None for auto-detection.
- **rows** (*int*) – Number of rows in the tileset.  
Can be left None for auto-detection.
- **columnFirst** (*bool*) – Defines if the character order goes along the rows or columns.  
It should be True if the character codes 0-15 are in the first column, and should be False if the characters 0-15 are in the first row.
- **greyscale** (*bool*) – Creates an anti-aliased font from a greyscale bitmap. Otherwise it uses the alpha channel for anti-aliasing.  
Unless you actually need anti-aliasing from a font you know uses a smooth greyscale channel you should leave this on False.
- **altLayout** (*bool*) – An alternative layout with space in the upper left corner. The column parameter is ignored if this is True, find examples of this layout in the *font/libtcod/* directory included with the python-tdl source.

**Raises** *TDLError* – Will be raised if no file is found at path or if auto-detection fails.

`tdl.init` (*width*, *height*, *title=None*, *fullscreen=False*, *renderer='SDL'*)

Start the main console with the given width and height and return the root console.

Call the console's drawing functions. Then remember to use `L{tdl.flush}` to make what's drawn visible on the console.

### Parameters

- **width** (*int*) – width of the root console (in tiles)
- **height** (*int*) – height of the root console (in tiles)
- **title** (*Optional[Text]*) – Text to display as the window title.  
If left None it defaults to the running script's filename.

- **fullscreen** (*bool*) – Can be set to True to start in fullscreen mode.
- **renderer** (*Text*) – Can be one of ‘GLSL’, ‘OPENGL’, or ‘SDL’.
- **to way Python works you're unlikely to see much of an** (*Due*) –
- **by using 'GLSL' over 'OPENGL' as most of the** (*improvement*) –
- **Python is slow interacting with the console and the** (*time*) –
- **itself is pretty fast even on 'SDL'.** (*rendering*) –

**Returns**

The root console.

Only what is drawn on the root console is what’s visible after a call to `tdl.flush`. After the root console is garbage collected, the window made by this function will close.

**Return type** `tdl.Console`

**See also:**

`Console.set_font`

`tdl.flush()`

Make all changes visible and update the screen.

Remember to call this function after drawing operations. Calls to flush will enforce the frame rate limit set by `tdl.set_fps`.

This function can only be called after `tdl.init`

`tdl.screenshot(path=None)`

Capture the screen and save it as a png file.

If path is None then the image will be placed in the current folder with the names: `screenshot001.png`, `screenshot002.png`, ...

**Parameters** `path` (*Optional[Text]*) – The file path to save the screenshot.

`tdl.get_fullscreen()`

Returns True if program is fullscreen.

**Returns**

**Returns True if the application is in full-screen mode.** Otherwise returns False.

**Return type** `bool`

`tdl.set_fullscreen(fullscreen)`

Changes the fullscreen state.

**Parameters** `fullscreen` (*bool*) – True for full-screen, False for windowed mode.

`tdl.set_title(title)`

Change the window title.

**Parameters** `title` (*Text*) – The new title text.

`tdl.get_fps()`

Return the current frames per second of the running program set by `set_fps`

**Returns**

**The frame rate set by `set_fps`.** If there is no current limit, this will return 0.

**Return type** `int`

`tdl.set_fps` (*fps*)

Set the maximum frame rate.

Further calls to `tdl.flush` will limit the speed of the program to run at *fps* frames per second. This can also be set to `None` to remove the frame rate limit.

**Parameters** `fps` (*optional[int]*) – The frames per second limit, or `None`.

`tdl.force_resolution` (*width, height*)

Change the fullscreen resolution.

**Parameters**

- **width** (*int*) – Width in pixels.
- **height** (*int*) – Height in pixels.

**exception** `tdl.TDLError`

The catch all for most TDL specific errors.

## 14.5 tdl.Console

**class** `tdl.Console` (*width, height*)

Contains character and color data and can be drawn to.

The console created by the `tdl.init` function is the root console and is the console that is rendered to the screen with `flush`.

Any console created from the `Console` class is an off-screen console that can be drawn on before being `blit` to the root console.

**Parameters**

- **width** (*int*) – Width of the new console in tiles
- **height** (*int*) – Height of the new console in tiles

`__contains__` (*position*)

Use `((x, y) in console)` to check if a position is drawable on this console.

`__del__` ()

If the main console is garbage collected then the window will be closed as well

`__iter__` ()

Return an iterator with every possible (x, y) value for this console.

It goes without saying that working on the console this way is a slow process, especially for Python, and should be minimized.

**Returns** An `((x, y), ...)` iterator.

**Return type** `Iterator[Tuple[int, int]]`

**blit** (*source, x=0, y=0, width=None, height=None, srcX=0, srcY=0, fg\_alpha=1.0, bg\_alpha=1.0*)

Blit another console or `Window` onto the current console.

By default it blits the entire source to the topleft corner.

**Parameters**

- **source** (`Union[tdl.Console, tdl.Window]`) – The blitting source. A console can blit to itself without any problems.
- **x** (*int*) – x-coordinate of this console to blit on.

- **y** (*int*) – y-coordinate of this console to blit on.
- **width** (*Optional[int]*) – Width of the rectangle.  
Can be None to extend as far as possible to the bottom right corner of the blit area or can be a negative number to be sized relative to the total size of the B{destination} console.
- **height** (*Optional[int]*) – Height of the rectangle.
- **srcX** (*int*) – x-coordinate of the source region to blit.
- **srcY** (*int*) – y-coordinate of the source region to blit.
- **fg\_alpha** (*float*) – The foreground alpha.

**clear** (*fg=Ellipsis, bg=Ellipsis*)

Clears the entire L{Console}/L{Window}.

Unlike other drawing functions, fg and bg can not be None.

#### Parameters

- **fg** (*Union[Tuple[int, int, int], int, Ellipsis]*)–
- **bg** (*Union[Tuple[int, int, int], int, Ellipsis]*)–

#### See also:

*draw\_rect*

**draw\_char** (*x, y, char, fg=Ellipsis, bg=Ellipsis*)

Draws a single character.

#### Parameters

- **x** (*int*) – x-coordinate to draw on.
- **y** (*int*) – y-coordinate to draw on.
- **char** (*Optional[Union[int, Text]]*) – An integer, single character string, or None.  
You can set the char parameter as None if you only want to change the colors of the tile.
- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*)–
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*)–

**Raises: AssertionError: Having x or y values that can't be placed** inside of the console will raise an AssertionError. You can use always use ((x, y) in console) to check if a tile is drawable.

#### See also:

*get\_char*

**draw\_frame** (*x, y, width, height, string, fg=Ellipsis, bg=Ellipsis*)

Similar to L{draw\_rect} but only draws the outline of the rectangle.

*width* or *height* can be None to extend to the bottom right of the console or can be a negative number to be sized relative to the total size of the console.

#### Parameters

- **x** (*int*) – The x-coordinate to start on.
- **y** (*int*) – The y-coordinate to start on.
- **width** (*Optional[int]*) – Width of the rectangle.

- **height** (*Optional[int]*) – Height of the rectangle.
- **string** (*Optional[Union[Text, int]]*) – An integer, single character string, or None.

You can set this parameter as None if you only want to change the colors of an area.

- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –

#### Raises

**AssertionError** – Having x or y values that can't be placed inside of the console will raise an AssertionError.

You can use always use ((x, y) in console) to check if a tile is drawable.

#### See also:

*draw\_rect, Window*

**draw\_rect** (x, y, width, height, string, fg=Ellipsis, bg=Ellipsis)

Draws a rectangle starting from x and y and extending to width and height.

If width or height are None then it will extend to the edge of the console.

#### Parameters

- **x** (*int*) – x-coordinate for the top side of the rect.
- **y** (*int*) – y-coordinate for the left side of the rect.
- **width** (*Optional[int]*) – The width of the rectangle.  
Can be None to extend to the bottom right of the console or can be a negative number to be sized relative to the total size of the console.
- **height** (*Optional[int]*) – The height of the rectangle.
- **string** (*Optional[Union[Text, int]]*) – An integer, single character string, or None.

You can set the string parameter as None if you only want to change the colors of an area.

- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –

#### Raises

- **AssertionError** – Having x or y values that can't be placed inside of the console will raise an AssertionError.
- You can use always use ((x, y) in console) to check if a tile
- is drawable.

#### See also:

*clear, draw\_frame*

**draw\_str** (x, y, string, fg=Ellipsis, bg=Ellipsis)

Draws a string starting at x and y.

A string that goes past the right side will wrap around. A string wrapping to below the console will raise `tdl.TDLError` but will still be written out. This means you can safely ignore the errors with a `try..except` block if you're fine with partially written strings.



r and n are drawn on the console as normal character tiles. No special encoding is done and any string will translate to the character table as is.

For a string drawing operation that respects special characters see `print_str`.

#### Parameters

- **x** (*int*) – x-coordinate to start at.
- **y** (*int*) – y-coordinate to start at.
- **string** (*Union[Text, Iterable[int]]*) – A string or an iterable of numbers.  
Special characters are ignored and rendered as any other character.
- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –

#### Raises

**AssertionError** – Having x or y values that can't be placed inside of the console will raise an AssertionError.

You can use always use `((x, y) in console)` to check if a tile is drawable.

#### See also:

`print_str`

**get\_char** (*x, y*)

Return the character and colors of a tile as (ch, fg, bg)

This method runs very slowly as is not recommended to be called frequently.

#### Parameters

- **x** (*int*) – The x-coordinate to pick.
- **y** (*int*) – The y-coordinate to pick.

#### Returns

A 3-item tuple: (*int, fg, bg*)

The first item is an integer of the character at the position (x, y) the second and third are the foreground and background colors respectively.

**Return type** `Tuple[int, Tuple[int, int, int], Tuple[int, int, int]]`

#### See also:

`draw_char`

**get\_cursor** ()

Return the virtual cursor position.

The cursor can be moved with the `move` method.

#### Returns

The (x, y) coordinate of where `print_str` will continue from.

**Return type** `Tuple[int, int]`

#### See also:

`:any:move`

**get\_size()**

Return the size of the console as (width, height)

**Returns** A (width, height) tuple.

**Return type** Tuple[int, int]

**move(x, y)**

Move the virtual cursor.

**Parameters**

- **x** (*int*) – x-coordinate to place the cursor.
- **y** (*int*) – y-coordinate to place the cursor.

**See also:**

*get\_cursor, print\_str, write*

**print\_str(string)**

Print a string at the virtual cursor.

Handles special characters such as ‘n’ and ‘r’. Printing past the bottom of the console will scroll everything upwards if *set\_mode* is set to ‘scroll’.

Colors can be set with *set\_colors* and the virtual cursor can be moved with *move*.

**Parameters** **string** (*Text*) – The text to print.

**See also:**

*draw\_str, move, set\_colors, set\_mode, write, Window*

**scroll(x, y)**

Scroll the contents of the console in the direction of x,y.

Uncovered areas will be cleared to the default background color. Does not move the virtual cursor.

**Parameters**

- **x** (*int*) – Distance to scroll along the x-axis.
- **y** (*int*) – Distance to scroll along the y-axis.

**Returns** An iterator over the (x, y) coordinates of any tile uncovered after scrolling.

**Return type** Iterator[Tuple[int, int]]

**See also:**

*set\_colors*

**set\_colors(fg=None, bg=None)**

Sets the colors to be used with the L{print\_str} and draw\_\* methods.

Values of None will only leave the current values unchanged.

**Parameters**

- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –

**See also:**

*move, print\_str*

**set\_mode** (*mode*)

Configure how this console will react to the cursor writing past the end of the console.

This is for methods that use the virtual cursor, such as `print_str`.

#### Parameters

- **mode** (*Text*) – The mode to set.
- **settings are** (*Possible*) –
  - ‘error’ - A `TDLError` will be raised once the cursor reaches the end of the console. Everything up until the error will still be drawn. This is the default setting.
  - ‘scroll’ - The console will scroll up as stuff is written to the end. You can restrict the region with `tdl.Window` when doing this.

..seealso: `write`, `print_str`

**write** (*string*)

This method mimics basic file-like behaviour.

Because of this method you can replace `sys.stdout` or `sys.stderr` with a `Console` or `Window` instance.

This is a convoluted process and behaviour seen now can be expected to change on later versions.

**Parameters** **string** (*Text*) – The text to write out.

**See also:**

`set_colors`, `set_mode`, `Window`

## 14.6 tdl.Window

**class** `tdl.Window` (*console, x, y, width, height*)

Isolate part of a `Console` or `Window` instance.

This classes methods are the same as `tdl.Console`

Making a `Window` and setting its width or height to `None` will extend it to the edge of the console.

This follows the normal rules for indexing so you can use a negative integer to place the `Window` relative to the bottom right of the parent `Console` instance.

*width* or *height* can be set to `None` to extend as far as possible to the bottom right corner of the parent `Console` or can be a negative number to be sized relative to the `Consoles` total size.

#### Parameters

- **console** (*Union*(`tdl.Console`, `tdl.Window`)) – The parent object.
- **x** (*int*) – x-coordinate to place the `Window`.
- **y** (*int*) – y-coordinate to place the `Window`.
- **width** (*Optional*[*int*]) – Width of the `Window`.
- **height** (*Optional*[*int*]) – Height of the `Window`.

**clear** (*fg=Ellipsis, bg=Ellipsis*)

Clears the entire L{`Console`}/L{`Window`}.

Unlike other drawing functions, `fg` and `bg` can not be `None`.

#### Parameters

- **fg** (*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*)]-
- **bg** (*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*)]-

See also:

*draw\_rect*

**draw\_char** (*x*, *y*, *char*, *fg=Ellipsis*, *bg=Ellipsis*)  
Draws a single character.

#### Parameters

- **x** (*int*) – x-coordinate to draw on.
- **y** (*int*) – y-coordinate to draw on.
- **char** (*Optional*[*Union*[*int*, *Text*]]) – An integer, single character string, or None.

You can set the char parameter as None if you only want to change the colors of the tile.

- **fg** (*Optional*[*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*]])-
- **bg** (*Optional*[*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*]])-

**Raises: AssertionError: Having x or y values that can't be placed** inside of the console will raise an AssertionError. You can use always use ((*x*, *y*) in console) to check if a tile is drawable.

See also:

*get\_char*

**draw\_frame** (*x*, *y*, *width*, *height*, *string*, *fg=Ellipsis*, *bg=Ellipsis*)  
Similar to L{draw\_rect} but only draws the outline of the rectangle.

*width* or *height* can be None to extend to the bottom right of the console or can be a negative number to be sized relative to the total size of the console.

#### Parameters

- **x** (*int*) – The x-coordinate to start on.
- **y** (*int*) – The y-coordinate to start on.
- **width** (*Optional*[*int*]) – Width of the rectangle.
- **height** (*Optional*[*int*]) – Height of the rectangle.
- **string** (*Optional*[*Union*[*Text*, *int*]]) – An integer, single character string, or None.

You can set this parameter as None if you only want to change the colors of an area.

- **fg** (*Optional*[*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*]])-
- **bg** (*Optional*[*Union*[*Tuple*[*int*, *int*, *int*], *int*, *Ellipsis*]])-

#### Raises

**AssertionError – Having x or y values that can't be placed inside** of the console will raise an AssertionError.

You can use always use ((*x*, *y*) in console) to check if a tile is drawable.

See also:

*draw\_rect*, *Window*

**draw\_rect** (*x*, *y*, *width*, *height*, *string*, *fg=Ellipsis*, *bg=Ellipsis*)

Draws a rectangle starting from *x* and *y* and extending to *width* and *height*.

If *width* or *height* are `None` then it will extend to the edge of the console.

#### Parameters

- **x** (*int*) – x-coordinate for the top side of the rect.
- **y** (*int*) – y-coordinate for the left side of the rect.
- **width** (*Optional[int]*) – The width of the rectangle.  
Can be `None` to extend to the bottom right of the console or can be a negative number to be sized relative to the total size of the console.
- **height** (*Optional[int]*) – The height of the rectangle.
- **string** (*Optional[Union[Text, int]]*) – An integer, single character string, or `None`.  
You can set the `string` parameter as `None` if you only want to change the colors of an area.
- **fg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –
- **bg** (*Optional[Union[Tuple[int, int, int], int, Ellipsis]]*) –

#### Raises

- `AssertionError` – Having *x* or *y* values that can't be placed inside of the console will raise an `AssertionError`.
- You can always use `((x, y) in console)` to check if a tile
- is drawable.

#### See also:

`clear`, `draw_frame`

**get\_char** (*x*, *y*)

Return the character and colors of a tile as (*ch*, *fg*, *bg*)

This method runs very slowly as is not recommended to be called frequently.

#### Parameters

- **x** (*int*) – The x-coordinate to pick.
- **y** (*int*) – The y-coordinate to pick.

#### Returns

A 3-item tuple: (*int*, *fg*, *bg*)

The first item is an integer of the character at the position (*x*, *y*) the second and third are the foreground and background colors respectively.

**Return type** `Tuple[int, Tuple[int, int, int], Tuple[int, int, int]]`

#### See also:

`draw_char`



This module handles user input.

To handle user input you will likely want to use the `event.get` function or create a subclass of `event.App`.

- `tdl.event.get` iterates over recent events.
- `tdl.event.App` passes events to the overridable methods: `ev_*` and `key_*`.

But there are other options such as `event.key_wait` and `event.is_window_closed`.

A few event attributes are actually string constants. Here's a reference for those:

- `Event.type`: 'QUIT', 'KEYDOWN', 'KEYUP', 'MOUSEDOWN', 'MOUSEUP', or 'MOUSEMOTION.'
- `MouseButtonEvent.button` (found in `MouseDown` and `MouseUp` events): 'LEFT', 'MIDDLE', 'RIGHT', 'SCROLLUP', 'SCROLLDOWN'
- `KeyEvent.key` (found in `KeyDown` and `KeyUp` events): 'NONE', 'ESCAPE', 'BACKSPACE', 'TAB', 'ENTER', 'SHIFT', 'CONTROL', 'ALT', 'PAUSE', 'CAPSLOCK', 'PAGEUP', 'PAGEDOWN', 'END', 'HOME', 'UP', 'LEFT', 'RIGHT', 'DOWN', 'PRINTSCREEN', 'INSERT', 'DELETE', 'LWIN', 'RWIN', 'APPS', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'KP0', 'KP1', 'KP2', 'KP3', 'KP4', 'KP5', 'KP6', 'KP7', 'KP8', 'KP9', 'KPADD', 'KPSUB', 'KPDIV', 'KPMUL', 'KPDEC', 'KPENTER', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12', 'NUMLOCK', 'SCROLLLOCK', 'SPACE', 'CHAR'

**class** `tdl.event.Event`

Base Event class.

You can easily subclass this to make your own events. Be sure to set the class attribute `L{Event.type}` for it to be passed to a custom `App ev_*` method.

`__repr__()`

List an events public attributes when printed.

`type = None`

String constant representing the type of event.

The `App ev_*` methods depend on this attribute.

Can be: 'QUIT', 'KEYDOWN', 'KEYUP', 'MOUSEDOWN', 'MOUSEUP', or 'MOUSEMOTION.'

**class** `tdl.event.Quit`

Fired when the window is closed by the user.

**class** `tdl.event.KeyEvent` (*key=""*, *char=""*, *text=""*, *shift=False*, *left\_alt=False*, *right\_alt=False*,  
*left\_control=False*, *right\_control=False*, *left\_meta=False*,  
*right\_meta=False*)

Base class for key events.

**alt = None**

True if alt was held down during this event.

**Type** `bool`

**char = None**

A single character string of the letter or symbol pressed.

Special characters like delete and return are not cross-platform. `L{key}` or `L{keychar}` should be used instead for special keys. Characters are also case sensitive.

**Type** `Text`

**control = None**

True if control was held down during this event.

**Type** `bool`

**key = None**

Human readable names of the key pressed. Non special characters will show up as 'CHAR'.

Can be one of 'NONE', 'ESCAPE', 'BACKSPACE', 'TAB', 'ENTER', 'SHIFT', 'CONTROL', 'ALT', 'PAUSE', 'CAPSLOCK', 'PAGEUP', 'PAGEDOWN', 'END', 'HOME', 'UP', 'LEFT', 'RIGHT', 'DOWN', 'PRINTSCREEN', 'INSERT', 'DELETE', 'LWIN', 'RWIN', 'APPS', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'KP0', 'KP1', 'KP2', 'KP3', 'KP4', 'KP5', 'KP6', 'KP7', 'KP8', 'KP9', 'KPADD', 'KPSUB', 'KPDIV', 'KPMUL', 'KPDEC', 'KPENTER', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12', 'NUMLOCK', 'SCROLLLOCK', 'SPACE', 'CHAR'

For the actual character instead of 'CHAR' use *keychar*.

**Type** `Text`

**keychar = None**

Similar to `L{key}` but returns a case sensitive letter or symbol instead of 'CHAR'.

This variable makes available the widest variety of symbols and should be used for key-mappings or anywhere where a narrower sample of keys isn't needed.

**left\_alt = None**

type: `bool`

**left\_control = None**

type: `bool`

**right\_alt = None**

type: `bool`

**right\_control = None**

type: `bool`

**shift = None**

True if shift was held down during this event.

**Type** `bool`



**class** `tdl.event.KeyDown` (*key=*”, *char=*”, *text=*”, *shift=False*, *left\_alt=False*, *right\_alt=False*,  
*left\_control=False*, *right\_control=False*, *left\_meta=False*,  
*right\_meta=False*)

Fired when the user presses a key on the keyboard or a key repeats.

**class** `tdl.event.KeyUp` (*key=*”, *char=*”, *text=*”, *shift=False*, *left\_alt=False*, *right\_alt=False*,  
*left\_control=False*, *right\_control=False*, *left\_meta=False*, *right\_meta=False*)

Fired when the user releases a key on the keyboard.

**class** `tdl.event.MouseButtonEvent` (*button*, *pos*, *cell*)

Base class for mouse button events.

**button = None**

Can be one of ‘LEFT’, ‘MIDDLE’, ‘RIGHT’, ‘SCROLLUP’, ‘SCROLLDOWN’

**Type** Text

**cell = None**

(x, y) position of the mouse snapped to a cell on the root console

**Type** Tuple[int, int]

**pos = None**

(x, y) position of the mouse on the screen.

**Type** Tuple[int, int]

**class** `tdl.event.MouseDown` (*button*, *pos*, *cell*)

Fired when a mouse button is pressed.

**class** `tdl.event.MouseUp` (*button*, *pos*, *cell*)

Fired when a mouse button is released.

**class** `tdl.event.MouseMotion` (*pos*, *cell*, *motion*, *cellmotion*)

Fired when the mouse is moved.

**cell = None**

(x, y) position of the mouse snapped to a cell on the root console. type: (int, int)

**cellmotion = None**

(x, y) motion of the mouse moving over cells on the root console. type: (int, int)

**motion = None**

(x, y) motion of the mouse on the screen. type: (int, int)

**pos = None**

(x, y) position of the mouse on the screen. type: (int, int)

**class** `tdl.event.App`

Application framework.

- `ev_*`: Events are passed to methods based on their `Event.type` attribute. If an event type is ‘KEY-DOWN’ the `ev_KEYDOWN` method will be called with the event instance as a parameter.
- `key_*`: When a key is pressed another method will be called based on the `KeyEvent.key` attribute. For example the ‘ENTER’ key will call `key_ENTER` with the associated `KeyDown` event as its parameter.
- `update`: This method is called every loop. It is passed a single parameter detailing the time in seconds since the last update (often known as `deltaTime`.)

You may want to call drawing routines in this method followed by `tdl.flush`.

**ev\_KEYDOWN** (*event*)

Override this method to handle a `KeyDown` event.

**ev\_KEYUP** (*event*)

Override this method to handle a *KeyUp* event.

**ev\_MOUSEDOWN** (*event*)

Override this method to handle a *MouseDown* event.

**ev\_MOUSEMOTION** (*event*)

Override this method to handle a *MouseMove* event.

**ev\_MOUSEUP** (*event*)

Override this method to handle a *MouseUp* event.

**ev\_QUIT** (*event*)

Unless overridden this method raises a `SystemExit` exception closing the program.

**run** ()

Delegate control over to this `App` instance. This function will process all events and send them to the special methods `ev_*` and `key_*`.

A call to `App.suspend` will return the control flow back to where this function is called. And then the `App` can be run again. But a single `App` instance can not be run multiple times simultaneously.

**run\_once** ()

Pump events to this `App` instance and then return.

This works in the way described in `App.run` except it immediately returns after the first `update` call.

Having multiple `App` instances and selectively calling `runOnce` on them is a decent way to create a state machine.

**suspend** ()

When called the `App` will begin to return control to where `App.run` was called.

Some further events are processed and the `App.update` method will be called one last time before exiting (unless suspended during a call to `App.update`.)

**update** (*deltaTime*)

Override this method to handle per frame logic and drawing.

**Parameters** `deltaTime` (*float*) – This parameter tells the amount of time passed since the last call measured in seconds as a floating point number.

You can use this variable to make your program frame rate independent. Use this parameter to adjust the speed of motion, timers, and other game logic.

`tdl.event.get` ()

Flushes the event queue and returns the list of events.

This function returns `Event` objects that can be identified by their `type` attribute or their class.

**Returns:** `Iterator[Type[Event]]`: An iterable of `Events` or anything put in a `push` call.

If the iterator is deleted or otherwise interrupted before finishing the excess items are preserved for the next call.

`tdl.event.wait` (*timeout=None, flush=True*)

Wait for an event.

**Parameters**

- **timeout** (*Optional[int]*) – The time in seconds that this function will wait before giving up and returning `None`.

With the default value of `None`, this will block forever.

- **flush** (*bool*) – If True a call to `tdl.flush` will be made before listening for events.

**Returns:** Type[Event]: An event, or None if the function has timed out. Anything added via `push` will also be returned.

`tdl.event.push(event)`

Push an event into the event buffer.

**Parameters** **event** (*Any*) – This event will be available on the next call to `event.get`.

An event pushed in the middle of a `get` will not show until the next time `get` called preventing push related infinite loops.

This object should at least have a 'type' attribute.

`tdl.event.key_wait()`

Waits until the user presses a key. Then returns a `KeyDown` event.

Key events will repeat if held down.

A click to close the window will be converted into an Alt+F4 `KeyDown` event.

**Returns** The pressed key.

**Return type** `tdl.event.KeyDown`

`tdl.event.set_key_repeat(delay=500, interval=0)`

Does nothing.

`tdl.event.is_window_closed()`

Returns True if the exit button on the window has been clicked and stays True afterwards.

Returns: bool:



Rogue-like map utilities such as line-of-sight, field-of-view, and path-finding.

Deprecated since version 3.2: The features provided here are better realized in the *tcod.map* and *tcod.path* modules.

**class** `tdl.map.AStar` (*width*, *height*, *callback*, *diagonalCost*=1.4142135623730951, *advanced*=False)  
An A\* pathfinder using a callback.

Deprecated since version 3.2: See *tcod.path*.

Before crating this instance you should make one of two types of callbacks:

- A function that returns the cost to move to (x, y)
- A function that returns the cost to move between (destX, destY, sourceX, sourceY)

If path is blocked the function should return zero or None. When using the second type of callback be sure to set *advanced*=True

#### Parameters

- **width** (*int*) – Width of the pathfinding area (in tiles.)
- **height** (*int*) – Height of the pathfinding area (in tiles.)
- **(Union[Callable[[int, int], float], (callback) – Callable[[int, int, int, int], float]]):** A callback returning the cost of a tile or edge.

A callback taking parameters depending on the setting of ‘advanced’ and returning the cost of movement for an open tile or zero for a blocked tile.

- **diagonalCost** (*float*) – Multiplier for diagonal movement.

Can be set to zero to disable diagonal movement entirely.

- **advanced** (*bool*) – Give 2 additional parameters to the callback.

A simple callback with 2 positional parameters may not provide enough information. Setting this to True will call the callback with 2 additional parameters giving you both the destination and the source of movement.

When True the callback will need to accept (destX, destY, sourceX, sourceY) as parameters. Instead of just (destX, destY).

**get\_path** (*origX, origY, destX, destY*)  
Get the shortest path from origXY to destXY.

#### Returns

Returns a list walking the path from **orig** to **dest**.

This excludes the starting point and includes the destination.

If no path is found then an empty list is returned.

**Return type** List[Tuple[int, int]]

**class** `tdl.map.Map` (*width, height, order='F'*)  
Field-of-view and path-finding on stored data.

Changed in version 4.1: *transparent*, *walkable*, and *fov* are now numpy boolean arrays.

Changed in version 4.3: Added *order* parameter.

Deprecated since version 3.2: `tcod.map.Map` should be used instead.

Set map conditions with the walkable and transparency attributes, this object can be iterated and checked for containment similar to consoles.

For example, you can set all tiles and transparent and walkable with the following code:

#### Example

```
>>> import tdl.map
>>> map_ = tdl.map.Map(80, 60)
>>> map_.transparent[:] = True
>>> map_.walkable[:] = True
```

#### **transparent**

Map transparency

Access this attribute with `map.transparent[x, y]`

Set to True to allow field-of-view rays, False will block field-of-view.

Transparent tiles only affect field-of-view.

#### **walkable**

Map accessibility

Access this attribute with `map.walkable[x, y]`

Set to True to allow path-finding through that tile, False will block passage to that tile.

Walkable tiles only affect path-finding.

#### **fov**

Map tiles touched by a field-of-view computation.

Access this attribute with `map.fov[x, y]`

Is True if a the tile is if view, otherwise False.

You can set to this attribute if you want, but you'll typically be using it to read the field-of-view of a `compute_fov` call.

**compute\_fov** (*x*, *y*, *fov*='PERMISSIVE', *radius*=None, *light\_walls*=True, *sphere*=True, *cumulative*=False)

Compute the field-of-view of this Map and return an iterator of the points touched.

#### Parameters

- **x** (*int*) – Point of view, x-coordinate.
- **y** (*int*) – Point of view, y-coordinate.
- **fov** (*Text*) – The type of field-of-view to be used.  
Available types are: 'BASIC', 'DIAMOND', 'SHADOW', 'RESTRICTIVE', 'PERMISSIVE', 'PERMISSIVE0', 'PERMISSIVE1', ..., 'PERMISSIVE8'
- **radius** (*Optional[int]*) – Maximum view distance from the point of view.  
A value of 0 will give an infinite distance.
- **light\_walls** (*bool*) – Light up walls, or only the floor.
- **sphere** (*bool*) – If True the lit area will be round instead of square.
- **cumulative** (*bool*) – If True the lit cells will accumulate instead of being cleared before the computation.

#### Returns

An iterator of (x, y) points of tiles touched by the field-of-view.

**Return type** Iterator[Tuple[int, int]]

**compute\_path** (*start\_x*, *start\_y*, *dest\_x*, *dest\_y*, *diagonal\_cost*=1.4142135623730951)

Get the shortest path between two points.

#### Parameters

- **start\_x** (*int*) – Starting x-position.
- **start\_y** (*int*) – Starting y-position.
- **dest\_x** (*int*) – Destination x-position.
- **dest\_y** (*int*) – Destination y-position.
- **diagonal\_cost** (*float*) – Multiplier for diagonal movement.  
Can be set to zero to disable diagonal movement entirely.

#### Returns

The shortest list of points to the destination position from the starting position.

The start point is not included in this list.

**Return type** List[Tuple[int, int]]

`tdl.map.bresenham` (*x1*, *y1*, *x2*, *y2*)

Return a list of points in a bresenham line.

Implementation hastily copied from RogueBasin.

#### Returns

A list of (x, y) points, including both the start and end-points.

**Return type** List[Tuple[int, int]]

`tdl.map.quick_fov(x, y, callback, fov='PERMISSIVE', radius=7.5, lightWalls=True, sphere=True)`  
All field-of-view functionality in one call.

Before using this call be sure to make a function, lambda, or method that takes 2 positional parameters and returns True if light can pass through the tile or False for light-blocking tiles and for indexes that are out of bounds of the dungeon.

This function is 'quick' as in no hassle but can quickly become a very slow function call if a large radius is used or the callback provided itself isn't optimized.

Always check if the index is in bounds both in the callback and in the returned values. These values can go into the negatives as well.

#### Parameters

- **x** (*int*) – x center of the field-of-view
- **y** (*int*) – y center of the field-of-view
- **callback** (*Callable[[int, int], bool]*) – This should be a function that takes two positional arguments x,y and returns True if the tile at that position is transparent or False if the tile blocks light or is out of bounds.
- **fov** (*Text*) – The type of field-of-view to be used.  
Available types are: 'BASIC', 'DIAMOND', 'SHADOW', 'RESTRICTIVE', 'PERMISSIVE', 'PERMISSIVE0', 'PERMISSIVE1', ..., 'PERMISSIVE8'
- **radius** (*float*) – When sphere is True a floating point can be used to fine-tune the range. Otherwise the radius is just rounded up.  
Be careful as a large radius has an exponential affect on how long this function takes.
- **lightWalls** (*bool*) – Include or exclude wall tiles in the field-of-view.
- **sphere** (*bool*) – True for a spherical field-of-view. False for a square one.

#### Returns

A set of (x, y) points that are within the field-of-view.

**Return type** Set[Tuple[int, int]]



This module provides advanced noise generation.

Noise is sometimes used for over-world generation, height-maps, and cloud/mist/smoke effects among other things.

You can see examples of the available noise algorithms in the libtcod documentation [here](#).

```
class tdl.noise.Noise (algorithm='PERLIN', mode='FLAT', hurst=0.5, lacunarity=2.0, octaves=4.0, seed=None, dimensions=4)
```

An advanced noise generator.

Deprecated since version 3.2: This class has been replaced by `tcod.noise.Noise`.

#### Parameters

- **algorithm** (*Text*) – The primary noise algorithm to be used.  
Can be one of 'PERLIN', 'SIMPLEX', 'WAVELET'
  - 'PERLIN' - A popular noise generator.
  - 'SIMPLEX' - In theory this is a slightly faster generator with less noticeable directional artifacts.
  - 'WAVELET' - A noise generator designed to reduce aliasing and not lose detail when summed into a fractal (as with the 'FBM' and 'TURBULENCE' modes.) This works faster at higher dimensions.
- **mode** (*Text*) – A secondary parameter to determine how noise is generated.  
Can be one of 'FLAT', 'FBM', 'TURBULENCE'
  - 'FLAT' - Generates the simplest form of noise. This mode does not use the hurst, lacunarity, and octaves parameters.
  - 'FBM' - Generates fractal brownian motion.
  - 'TURBULENCE' - Generates detailed noise with smoother and more natural transitions.
- **hurst** (*float*) – The hurst exponent.

This describes the raggedness of the resultant noise, with a higher value leading to a smoother noise. It should be in the 0.0-1.0 range.

This is only used in 'FBM' and 'TURBULENCE' modes.

- **lacunarity** (*float*) – A multiplier that determines how quickly the frequency increases for each successive octave.

The frequency of each successive octave is equal to the product of the previous octave's frequency and the lacunarity value.

This is only used in 'FBM' and 'TURBULENCE' modes.

- **octaves** (*float*) – Controls the amount of detail in the noise.

This is only used in 'FBM' and 'TURBULENCE' modes.

- **seed** (*Hashable*) – You can use any hashable object to be a seed for the noise generator.

If None is used then a random seed will be generated.

**get\_point** (*\*position*)

Return the noise value of a specific position.

Example usage: `value = noise.getPoint(x, y, z)`

**Parameters** **position** (*Tuple[`float`, ...]*) – The point to sample at.

**Returns**

The noise value at position.

This will be a floating point in the 0.0-1.0 range.

**Return type** `float`

## CHAPTER 18

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**t**

tcod, ??  
tcod.bsp, 29  
tcod.console, 33  
tcod.event, 43  
tcod.image, 51  
tcod.map, 55  
tcod.noise, 59  
tcod.path, 63  
tcod.random, 67  
tcod.tileset, 69  
tdl, 109  
tdl.event, 121  
tdl.map, 127  
tdl.noise, 131



## Symbols

\_\_bool\_\_() (*tcod.console.Console method*), 34  
 \_\_contains\_\_() (*tcod.tileset.Tileset method*), 69  
 \_\_contains\_\_() (*tdl.Console method*), 112  
 \_\_del\_\_() (*tdl.Console method*), 112  
 \_\_enter\_\_() (*tcod.console.Console method*), 34  
 \_\_eq\_\_() (*tcod.Color method*), 73  
 \_\_exit\_\_() (*tcod.console.Console method*), 34  
 \_\_getitem\_\_() (*tcod.Color method*), 73  
 \_\_getstate\_\_() (*tcod.random.Random method*), 67  
 \_\_iter\_\_() (*tdl.Console method*), 112  
 \_\_new\_\_() (*tcod.path.NodeCostArray static method*), 64  
 \_\_repr\_\_() (*tcod.Color method*), 73  
 \_\_repr\_\_() (*tcod.Key method*), 84  
 \_\_repr\_\_() (*tcod.Mouse method*), 86  
 \_\_repr\_\_() (*tcod.console.Console method*), 34  
 \_\_repr\_\_() (*tdl.event.Event method*), 121  
 \_\_setstate\_\_() (*tcod.random.Random method*), 67  
 \_\_str\_\_() (*tcod.bsp.BSP method*), 30  
 \_\_str\_\_() (*tcod.console.Console method*), 34

## A

alt (*tdl.event.KeyEvent attribute*), 122  
 App (*class in tdl.event*), 123  
 AStar (*class in tcod.path*), 64  
 AStar (*class in tdl.map*), 127

## B

b (*tcod.Color attribute*), 73  
 bg (*tcod.console.Console attribute*), 39  
 blit() (*tcod.console.Console method*), 34  
 blit() (*tcod.ConsoleBuffer method*), 106  
 blit() (*tcod.image.Image method*), 51  
 blit() (*tdl.Console method*), 112  
 blit\_2x() (*tcod.image.Image method*), 51  
 blit\_rect() (*tcod.image.Image method*), 52  
 bresenham() (*in module tdl.map*), 129  
 BSP (*class in tcod.bsp*), 29

bsp\_contains() (*in module tcod*), 72  
 bsp\_delete() (*in module tcod*), 72  
 bsp\_father() (*in module tcod*), 71  
 bsp\_find\_node() (*in module tcod*), 72  
 bsp\_is\_leaf() (*in module tcod*), 72  
 bsp\_left() (*in module tcod*), 71  
 bsp\_new\_with\_size() (*in module tcod*), 71  
 bsp\_remove\_sons() (*in module tcod*), 72  
 bsp\_resize() (*in module tcod*), 71  
 bsp\_right() (*in module tcod*), 71  
 bsp\_split\_once() (*in module tcod*), 71  
 bsp\_split\_recursive() (*in module tcod*), 71  
 bsp\_traverse\_in\_order() (*in module tcod*), 72  
 bsp\_traverse\_inverted\_level\_order() (*in module tcod*), 72  
 bsp\_traverse\_level\_order() (*in module tcod*), 72  
 bsp\_traverse\_post\_order() (*in module tcod*), 72  
 bsp\_traverse\_pre\_order() (*in module tcod*), 72  
 button (*tcod.event.MouseButtonEvent attribute*), 46  
 button (*tdl.event.MouseButtonEvent attribute*), 123

## C

c (*tcod.Key attribute*), 84  
 cell (*tdl.event.MouseButtonEvent attribute*), 123  
 cell (*tdl.event.MouseMotion attribute*), 123  
 cellmotion (*tdl.event.MouseMotion attribute*), 123  
 ch (*tcod.console.Console attribute*), 39  
 char (*tdl.event.KeyEvent attribute*), 122  
 children (*tcod.bsp.BSP attribute*), 30  
 clear() (*tcod.console.Console method*), 35  
 clear() (*tcod.ConsoleBuffer method*), 106  
 clear() (*tcod.image.Image method*), 52  
 clear() (*tdl.Console method*), 113  
 clear() (*tdl.Window method*), 117  
 Color (*class in tcod*), 72  
 color\_gen\_map() (*in module tcod*), 74  
 color\_get\_hsv() (*in module tcod*), 74  
 color\_lerp() (*in module tcod*), 73

- color\_scale\_HSV() (in module *tcod*), 74  
 color\_set\_hsv() (in module *tcod*), 73  
 compute\_fov() (in module *tcod.map*), 56  
 compute\_fov() (*tcod.map.Map* method), 56  
 compute\_fov() (*tdl.map.Map* method), 128  
 compute\_path() (*tdl.map.Map* method), 129  
 Console (class in *tcod.console*), 33  
 Console (class in *tdl*), 112  
 console defaults, 7  
 console\_blit() (in module *tcod*), 76  
 console\_c (*tcod.console.Console* attribute), 34  
 console\_check\_for\_keypress() (in module *tcod*), 76  
 console\_clear() (in module *tcod*), 76  
 console\_credits() (in module *tcod*), 76  
 console\_credits\_render() (in module *tcod*), 76  
 console\_credits\_reset() (in module *tcod*), 76  
 console\_delete() (in module *tcod*), 76  
 console\_fill\_background() (in module *tcod*), 76  
 console\_fill\_char() (in module *tcod*), 76  
 console\_fill\_foreground() (in module *tcod*), 77  
 console\_flush() (in module *tcod*), 76  
 console\_from\_file() (in module *tcod*), 77  
 console\_from\_xp() (in module *tcod*), 77  
 console\_get\_alignment() (in module *tcod*), 77  
 console\_get\_background\_flag() (in module *tcod*), 77  
 console\_get\_char() (in module *tcod*), 77  
 console\_get\_char\_background() (in module *tcod*), 77  
 console\_get\_char\_foreground() (in module *tcod*), 77  
 console\_get\_default\_background() (in module *tcod*), 77  
 console\_get\_default\_foreground() (in module *tcod*), 77  
 console\_get\_fade() (in module *tcod*), 78  
 console\_get\_fading\_color() (in module *tcod*), 78  
 console\_get\_height() (in module *tcod*), 78  
 console\_get\_height\_rect() (in module *tcod*), 78  
 console\_get\_width() (in module *tcod*), 78  
 console\_hline() (in module *tcod*), 78  
 console\_init\_root() (in module *tcod*), 75  
 console\_is\_fullscreen() (in module *tcod*), 78  
 console\_is\_key\_pressed() (in module *tcod*), 78  
 console\_is\_window\_closed() (in module *tcod*), 78  
 console\_list\_load\_xp() (in module *tcod*), 79  
 console\_list\_save\_xp() (in module *tcod*), 79  
 console\_load\_apf() (in module *tcod*), 78  
 console\_load\_asc() (in module *tcod*), 78  
 console\_load\_xp() (in module *tcod*), 78  
 console\_map\_ascii\_code\_to\_font() (in module *tcod*), 79  
 console\_map\_ascii\_codes\_to\_font() (in module *tcod*), 79  
 console\_map\_string\_to\_font() (in module *tcod*), 79  
 console\_new() (in module *tcod*), 79  
 console\_print() (in module *tcod*), 79  
 console\_print\_ex() (in module *tcod*), 80  
 console\_print\_frame() (in module *tcod*), 80  
 console\_print\_rect() (in module *tcod*), 80  
 console\_print\_rect\_ex() (in module *tcod*), 80  
 console\_put\_char() (in module *tcod*), 80  
 console\_put\_char\_ex() (in module *tcod*), 81  
 console\_rect() (in module *tcod*), 81  
 console\_save\_apf() (in module *tcod*), 81  
 console\_save\_asc() (in module *tcod*), 81  
 console\_save\_xp() (in module *tcod*), 81  
 console\_set\_alignment() (in module *tcod*), 81  
 console\_set\_background\_flag() (in module *tcod*), 81  
 console\_set\_char() (in module *tcod*), 82  
 console\_set\_char\_background() (in module *tcod*), 82  
 console\_set\_char\_foreground() (in module *tcod*), 82  
 console\_set\_color\_control() (in module *tcod*), 82  
 console\_set\_custom\_font() (in module *tcod*), 75  
 console\_set\_default\_background() (in module *tcod*), 82  
 console\_set\_default\_foreground() (in module *tcod*), 83  
 console\_set\_fade() (in module *tcod*), 83  
 console\_set\_fullscreen() (in module *tcod*), 83  
 console\_set\_key\_color() (in module *tcod*), 83  
 console\_set\_window\_title() (in module *tcod*), 83  
 console\_vline() (in module *tcod*), 83  
 console\_wait\_for\_keypress() (in module *tcod*), 83  
 ConsoleBuffer (class in *tcod*), 106  
 contains() (*tcod.bsp.BSP* method), 30  
 control (*tdl.event.KeyEvent* attribute), 122  
 copy() (*tcod.ConsoleBuffer* method), 107  
 cx (*tcod.Mouse* attribute), 85  
 cy (*tcod.Mouse* attribute), 85
- ## D
- dcx (*tcod.Mouse* attribute), 85  
 dcy (*tcod.Mouse* attribute), 85



- default\_alignment (*tcod.console.Console* attribute), 40
- default\_bg (*tcod.console.Console* attribute), 40
- default\_bg\_blend (*tcod.console.Console* attribute), 40
- default\_fg (*tcod.console.Console* attribute), 40
- Dice (*class in tcod*), 108
- Dijkstra (*class in tcod.path*), 64
- dijkstra2d() (*in module tcod.path*), 64
- dijkstra\_compute() (*in module tcod*), 89
- dijkstra\_delete() (*in module tcod*), 89
- dijkstra\_get() (*in module tcod*), 89
- dijkstra\_get\_distance() (*in module tcod*), 89
- dijkstra\_is\_empty() (*in module tcod*), 89
- dijkstra\_new() (*in module tcod*), 89
- dijkstra\_new\_using\_function() (*in module tcod*), 89
- dijkstra\_path\_set() (*in module tcod*), 89
- dijkstra\_path\_walk() (*in module tcod*), 89
- dijkstra\_reverse() (*in module tcod*), 89
- dijkstra\_size() (*in module tcod*), 89
- dispatch() (*tcod.event.EventDispatch* method), 48
- draw\_char() (*tdl.Console* method), 113
- draw\_char() (*tdl.Window* method), 118
- draw\_frame() (*tcod.console.Console* method), 35
- draw\_frame() (*tdl.Console* method), 113
- draw\_frame() (*tdl.Window* method), 118
- draw\_rect() (*tcod.console.Console* method), 36
- draw\_rect() (*tdl.Console* method), 114
- draw\_rect() (*tdl.Window* method), 118
- draw\_str() (*tdl.Console* method), 114
- DTYPE (*tcod.console.Console* attribute), 34
- dx (*tcod.Mouse* attribute), 85
- dy (*tcod.Mouse* attribute), 85
- ## E
- EdgeCostCallback (*class in tcod.path*), 64
- ev\_keydown() (*tcod.event.EventDispatch* method), 48
- ev\_KEYDOWN() (*tdl.event.App* method), 123
- ev\_keyup() (*tcod.event.EventDispatch* method), 48
- ev\_KEYUP() (*tdl.event.App* method), 123
- ev\_mousebuttondown() (*tcod.event.EventDispatch* method), 48
- ev\_mousebuttonup() (*tcod.event.EventDispatch* method), 48
- ev\_MOUSEBUTTONDOWN() (*tdl.event.App* method), 124
- ev\_mousemotion() (*tcod.event.EventDispatch* method), 48
- ev\_MOUSEMOTION() (*tdl.event.App* method), 124
- ev\_MOUSEUP() (*tdl.event.App* method), 124
- ev\_mousewheel() (*tcod.event.EventDispatch* method), 48
- ev\_quit() (*tcod.event.EventDispatch* method), 49
- ev\_QUIT() (*tdl.event.App* method), 124
- ev\_textinput() (*tcod.event.EventDispatch* method), 49
- ev\_windowclose() (*tcod.event.EventDispatch* method), 49
- ev\_windowenter() (*tcod.event.EventDispatch* method), 49
- ev\_windowexposed() (*tcod.event.EventDispatch* method), 49
- ev\_windowfocusgained() (*tcod.event.EventDispatch* method), 49
- ev\_windowfocuslost() (*tcod.event.EventDispatch* method), 49
- ev\_windowhidden() (*tcod.event.EventDispatch* method), 49
- ev\_windowleave() (*tcod.event.EventDispatch* method), 49
- ev\_windowmaximized() (*tcod.event.EventDispatch* method), 49
- ev\_windowminimized() (*tcod.event.EventDispatch* method), 49
- ev\_windowmoved() (*tcod.event.EventDispatch* method), 49
- ev\_windowresized() (*tcod.event.EventDispatch* method), 49
- ev\_windowrestored() (*tcod.event.EventDispatch* method), 49
- ev\_windowshown() (*tcod.event.EventDispatch* method), 49
- ev\_windowsizechanged() (*tcod.event.EventDispatch* method), 49
- Event (*class in tcod.event*), 43
- Event (*class in tdl.event*), 121
- EventDispatch (*class in tcod.event*), 48
- ## F
- fg (*tcod.console.Console* attribute), 40
- find\_node() (*tcod.bsp.BSP* method), 30
- flipped (*tcod.event.MouseWheel* attribute), 46
- flush() (*in module tdl*), 111
- force\_resolution() (*in module tdl*), 112
- fov (*tcod.map.Map* attribute), 55
- fov (*tdl.map.Map* attribute), 128
- from\_sdl\_event() (*tcod.event.Event* class method), 43
- from\_sdl\_event() (*tcod.event.KeyboardEvent* class method), 45
- from\_sdl\_event() (*tcod.event.MouseButtonEvent* class method), 46
- from\_sdl\_event() (*tcod.event.MouseMotion* class method), 45
- from\_sdl\_event() (*tcod.event.MouseWheel* class method), 47
- from\_sdl\_event() (*tcod.event.Quit* class method), 44

from\_sdl\_event() (*tcod.event.TextInput class method*), 47  
 from\_sdl\_event() (*tcod.event.Undefined class method*), 48  
 from\_sdl\_event() (*tcod.event.WindowEvent class method*), 47

## G

g (*tcod.Color attribute*), 73  
 get() (*in module tcod.event*), 49  
 get() (*in module tdl.event*), 124  
 get\_alpha() (*tcod.image.Image method*), 52  
 get\_char() (*tdl.Console method*), 115  
 get\_char() (*tdl.Window method*), 119  
 get\_cursor() (*tdl.Console method*), 115  
 get\_default() (*in module tcod.tileset*), 69  
 get\_fps() (*in module tdl*), 111  
 get\_fullscreen() (*in module tdl*), 111  
 get\_height\_rect() (*in module tcod.console*), 41  
 get\_height\_rect() (*tcod.console.Console method*), 36  
 get\_mipmap\_pixel() (*tcod.image.Image method*), 52  
 get\_path() (*tcod.path.AStar method*), 64  
 get\_path() (*tcod.path.Dijkstra method*), 64  
 get\_path() (*tdl.map.AStar method*), 128  
 get\_pixel() (*tcod.image.Image method*), 53  
 get\_point() (*tcod.noise.Noise method*), 60  
 get\_point() (*tdl.noise.Noise method*), 132  
 get\_size() (*tdl.Console method*), 115  
 get\_tile() (*tcod.tileset.Tileset method*), 69  
 gauss() (*tcod.random.Random method*), 67

## H

height (*tcod.bsp.BSP attribute*), 30  
 height (*tcod.console.Console attribute*), 40  
 height (*tcod.event.WindowResized attribute*), 47  
 height (*tcod.image.Image attribute*), 51  
 height (*tcod.map.Map attribute*), 55  
 heightmap\_add() (*in module tcod*), 91  
 heightmap\_add\_fbm() (*in module tcod*), 91  
 heightmap\_add\_hill() (*in module tcod*), 92  
 heightmap\_add\_hm() (*in module tcod*), 92  
 heightmap\_add\_voronoi() (*in module tcod*), 92  
 heightmap\_clamp() (*in module tcod*), 92  
 heightmap\_clear() (*in module tcod*), 93  
 heightmap\_copy() (*in module tcod*), 93  
 heightmap\_count\_cells() (*in module tcod*), 93  
 heightmap\_delete() (*in module tcod*), 93  
 heightmap\_dig\_bezier() (*in module tcod*), 93  
 heightmap\_dig\_hill() (*in module tcod*), 94  
 heightmap\_get\_interpolated\_value() (*in module tcod*), 94  
 heightmap\_get\_minmax() (*in module tcod*), 94

heightmap\_get\_normal() (*in module tcod*), 94  
 heightmap\_get\_slope() (*in module tcod*), 94  
 heightmap\_get\_value() (*in module tcod*), 95  
 heightmap\_has\_land\_on\_border() (*in module tcod*), 95  
 heightmap\_kernel\_transform() (*in module tcod*), 95  
 heightmap\_lerp\_hm() (*in module tcod*), 96  
 heightmap\_multiply\_hm() (*in module tcod*), 96  
 heightmap\_new() (*in module tcod*), 96  
 heightmap\_normalize() (*in module tcod*), 96  
 heightmap\_rain\_erosion() (*in module tcod*), 96  
 heightmap\_scale() (*in module tcod*), 97  
 heightmap\_scale\_fbm() (*in module tcod*), 97  
 heightmap\_set\_value() (*in module tcod*), 97  
 hflip() (*tcod.image.Image method*), 53  
 hillclimb2d() (*in module tcod.path*), 65  
 hline() (*tcod.console.Console method*), 36  
 horizontal (*tcod.bsp.BSP attribute*), 30

## I

Image (*class in tcod.image*), 51  
 image\_blit() (*in module tcod*), 98  
 image\_blit\_2x() (*in module tcod*), 98  
 image\_blit\_rect() (*in module tcod*), 98  
 image\_clear() (*in module tcod*), 98  
 image\_delete() (*in module tcod*), 98  
 image\_from\_console() (*in module tcod*), 98  
 image\_get\_alpha() (*in module tcod*), 98  
 image\_get\_mipmap\_pixel() (*in module tcod*), 98  
 image\_get\_pixel() (*in module tcod*), 98  
 image\_get\_size() (*in module tcod*), 98  
 image\_hflip() (*in module tcod*), 98  
 image\_invert() (*in module tcod*), 98  
 image\_is\_pixel\_transparent() (*in module tcod*), 98  
 image\_load() (*in module tcod*), 97  
 image\_new() (*in module tcod*), 98  
 image\_put\_pixel() (*in module tcod*), 98  
 image\_refresh\_console() (*in module tcod*), 98  
 image\_rotate90() (*in module tcod*), 98  
 image\_save() (*in module tcod*), 98  
 image\_scale() (*in module tcod*), 98  
 image\_set\_key\_color() (*in module tcod*), 98  
 image\_vflip() (*in module tcod*), 98  
 in\_order() (*tcod.bsp.BSP method*), 31  
 init() (*in module tdl*), 110  
 inverse\_gauss() (*tcod.random.Random method*), 68  
 invert() (*tcod.image.Image method*), 53  
 inverted\_level\_order() (*tcod.bsp.BSP method*), 31  
 is\_window\_closed() (*in module tdl.event*), 125

## K

Key (class in *tcod*), 84  
 key (*tdl.event.KeyEvent* attribute), 122  
 key\_wait () (in module *tdl.event*), 125  
 KeyboardEvent (class in *tcod.event*), 44  
 keychar (*tdl.event.KeyEvent* attribute), 122  
 KeyDown (class in *tcod.event*), 45  
 KeyDown (class in *tdl.event*), 122  
 KeyEvent (class in *tdl.event*), 122  
 KeyUp (class in *tcod.event*), 45  
 KeyUp (class in *tdl.event*), 123

## L

lalt (*tcod.Key* attribute), 84  
 lbutton (*tcod.Mouse* attribute), 85  
 lbutton\_pressed (*tcod.Mouse* attribute), 85  
 lctrl (*tcod.Key* attribute), 84  
 left\_alt (*tdl.event.KeyEvent* attribute), 122  
 left\_control (*tdl.event.KeyEvent* attribute), 122  
 level (*tcod.bsp.BSP* attribute), 30  
 level\_order () (*tcod.bsp.BSP* method), 31  
 libtcod-cffi, 7  
 libtcodpy, 7  
 line () (in module *tcod*), 99  
 line\_init () (in module *tcod*), 98  
 line\_iter () (in module *tcod*), 99  
 line\_step () (in module *tcod*), 99  
 line\_where () (in module *tcod*), 99  
 lmeta (*tcod.Key* attribute), 84  
 load\_truetype\_font () (in module *tcod.tileset*), 70

## M

Map (class in *tcod.map*), 55  
 Map (class in *tdl.map*), 128  
 map\_clear () (in module *tcod*), 100  
 map\_compute\_fov () (in module *tcod*), 100  
 map\_copy () (in module *tcod*), 100  
 map\_delete () (in module *tcod*), 100  
 map\_get\_height () (in module *tcod*), 100  
 map\_get\_width () (in module *tcod*), 100  
 map\_is\_in\_fov () (in module *tcod*), 100  
 map\_is\_transparent () (in module *tcod*), 101  
 map\_is\_walkable () (in module *tcod*), 101  
 map\_new () (in module *tcod*), 101  
 map\_set\_properties () (in module *tcod*), 101  
 maxarray () (in module *tcod.path*), 66  
 mbutton (*tcod.Mouse* attribute), 85  
 mbutton\_pressed (*tcod.Mouse* attribute), 85  
 mod (*tcod.event.KeyboardEvent* attribute), 44  
 motion (*tdl.event.MouseMotion* attribute), 123  
 Mouse (class in *tcod*), 84  
 mouse\_get\_status () (in module *tcod*), 101  
 mouse\_is\_cursor\_visible () (in module *tcod*), 101

mouse\_move () (in module *tcod*), 101  
 mouse\_show\_cursor () (in module *tcod*), 101  
 MouseButtonDown (class in *tcod.event*), 46  
 MouseButtonEvent (class in *tcod.event*), 45  
 MouseButtonEvent (class in *tdl.event*), 123  
 MouseButtonUp (class in *tcod.event*), 46  
 MouseDown (class in *tdl.event*), 123  
 MouseMotion (class in *tcod.event*), 45  
 MouseMotion (class in *tdl.event*), 123  
 MouseUp (class in *tdl.event*), 123  
 MouseWheel (class in *tcod.event*), 46  
 move () (*tdl.Console* method), 116

## N

namegen\_destroy () (in module *tcod*), 101  
 namegen\_generate () (in module *tcod*), 101  
 namegen\_generate\_custom () (in module *tcod*), 102  
 namegen\_get\_sets () (in module *tcod*), 102  
 namegen\_parse () (in module *tcod*), 102  
 NodeCostArray (class in *tcod.path*), 64  
 Noise (class in *tcod.noise*), 59  
 Noise (class in *tdl.noise*), 131  
 noise\_c (*tcod.noise.Noise* attribute), 60  
 noise\_delete () (in module *tcod*), 102  
 noise\_get () (in module *tcod*), 102  
 noise\_get\_fbm () (in module *tcod*), 102  
 noise\_get\_turbulence () (in module *tcod*), 102  
 noise\_new () (in module *tcod*), 102  
 noise\_set\_type () (in module *tcod*), 103

## P

parent (*tcod.bsp.BSP* attribute), 30  
 parser\_delete () (in module *tcod*), 103  
 parser\_get\_bool\_property () (in module *tcod*), 103  
 parser\_get\_char\_property () (in module *tcod*), 103  
 parser\_get\_color\_property () (in module *tcod*), 103  
 parser\_get\_dice\_property () (in module *tcod*), 103  
 parser\_get\_float\_property () (in module *tcod*), 103  
 parser\_get\_int\_property () (in module *tcod*), 103  
 parser\_get\_list\_property () (in module *tcod*), 103  
 parser\_get\_string\_property () (in module *tcod*), 103  
 parser\_new () (in module *tcod*), 103  
 parser\_new\_struct () (in module *tcod*), 103  
 parser\_run () (in module *tcod*), 103  
 path\_compute () (in module *tcod*), 89

`path_delete()` (in module *tcod*), 90  
`path_get()` (in module *tcod*), 90  
`path_get_destination()` (in module *tcod*), 90  
`path_get_origin()` (in module *tcod*), 90  
`path_is_empty()` (in module *tcod*), 90  
`path_new_using_function()` (in module *tcod*), 90  
`path_new_using_map()` (in module *tcod*), 90  
`path_reverse()` (in module *tcod*), 91  
`path_size()` (in module *tcod*), 91  
`path_walk()` (in module *tcod*), 91  
`pixel` (*tcod.event.MouseButtonEvent* attribute), 46  
`pixel` (*tcod.event.MouseMotion* attribute), 45  
`pixel_motion` (*tcod.event.MouseMotion* attribute), 45  
`Point` (class in *tcod.event*), 43  
`pos` (*tdl.event.MouseButtonEvent* attribute), 123  
`pos` (*tdl.event.MouseMotion* attribute), 123  
`position` (*tcod.bsp.BSP* attribute), 30  
`post_order()` (*tcod.bsp.BSP* method), 31  
`pre_order()` (*tcod.bsp.BSP* method), 31  
`pressed` (*tcod.Key* attribute), 84  
`print()` (*tcod.console.Console* method), 36  
`print_()` (*tcod.console.Console* method), 37  
`print_box()` (*tcod.console.Console* method), 37  
`print_frame()` (*tcod.console.Console* method), 37  
`print_rect()` (*tcod.console.Console* method), 38  
`print_str()` (*tdl.Console* method), 116  
`push()` (in module *tdl.event*), 125  
`put_char()` (*tcod.console.Console* method), 38  
`put_pixel()` (*tcod.image.Image* method), 53  
`python-tcod`, 7  
`python-tdl`, 7

## Q

`quick_fov()` (in module *tdl.map*), 129  
`Quit` (class in *tcod.event*), 44  
`Quit` (class in *tdl.event*), 121

## R

`r` (*tcod.Color* attribute), 73  
`ralt` (*tcod.Key* attribute), 84  
`randint()` (*tcod.random.Random* method), 68  
`Random` (class in *tcod.random*), 67  
`random_c` (*tcod.random.Random* attribute), 67  
`random_delete()` (in module *tcod*), 103  
`random_get_double()` (in module *tcod*), 103  
`random_get_double_mean()` (in module *tcod*), 104  
`random_get_float()` (in module *tcod*), 104  
`random_get_float_mean()` (in module *tcod*), 104  
`random_get_instance()` (in module *tcod*), 104  
`random_get_int()` (in module *tcod*), 104  
`random_get_int_mean()` (in module *tcod*), 105

`random_new()` (in module *tcod*), 105  
`random_new_from_seed()` (in module *tcod*), 105  
`random_restore()` (in module *tcod*), 105  
`random_save()` (in module *tcod*), 105  
`random_set_distribution()` (in module *tcod*), 105  
`rbutton` (*tcod.Mouse* attribute), 85  
`rbutton_pressed` (*tcod.Mouse* attribute), 85  
`rctrl` (*tcod.Key* attribute), 84  
`rect()` (*tcod.console.Console* method), 39  
`refresh_console()` (*tcod.image.Image* method), 53  
`repeat` (*tcod.event.KeyboardEvent* attribute), 45  
`right_alt` (*tdl.event.KeyEvent* attribute), 122  
`right_control` (*tdl.event.KeyEvent* attribute), 122  
`rmeta` (*tcod.Key* attribute), 84  
`rotate90()` (*tcod.image.Image* method), 53  
`run()` (*tdl.event.App* method), 124  
`run_once()` (*tdl.event.App* method), 124

## S

`sample_mgrid()` (*tcod.noise.Noise* method), 60  
`sample_ogrid()` (*tcod.noise.Noise* method), 60  
`save_as()` (*tcod.image.Image* method), 53  
`scale()` (*tcod.image.Image* method), 53  
`scancode` (*tcod.event.KeyboardEvent* attribute), 44  
`screenshot()` (in module *tdl*), 111  
`scroll()` (*tdl.Console* method), 116  
`sdl_event` (*tcod.event.Event* attribute), 43  
`set()` (*tcod.ConsoleBuffer* method), 107  
`set_back()` (*tcod.ConsoleBuffer* method), 107  
`set_colors()` (*tdl.Console* method), 116  
`set_default()` (in module *tcod.tileset*), 70  
`set_font()` (in module *tdl*), 110  
`set_fore()` (*tcod.ConsoleBuffer* method), 107  
`set_fps()` (in module *tdl*), 111  
`set_fullscreen()` (in module *tdl*), 111  
`set_goal()` (*tcod.path.Dijkstra* method), 64  
`set_key_color()` (*tcod.console.Console* method), 39  
`set_key_color()` (*tcod.image.Image* method), 53  
`set_key_repeat()` (in module *tdl.event*), 125  
`set_mode()` (*tdl.Console* method), 116  
`set_tile()` (*tcod.tileset.Tileset* method), 69  
`set_title()` (in module *tdl*), 111  
`set_truetype_font()` (in module *tcod.tileset*), 70  
`shift` (*tcod.Key* attribute), 84  
`shift` (*tdl.event.KeyEvent* attribute), 122  
`split_once()` (*tcod.bsp.BSP* method), 31  
`split_recursive()` (*tcod.bsp.BSP* method), 31  
`state` (*tcod.event.MouseMotion* attribute), 45  
`struct_add_flag()` (in module *tcod*), 106  
`struct_add_list_property()` (in module *tcod*), 106  
`struct_add_property()` (in module *tcod*), 106

struct\_add\_structure() (in module *tcod*), 106  
 struct\_add\_value\_list() (in module *tcod*), 106  
 struct\_get\_name() (in module *tcod*), 106  
 struct\_get\_type() (in module *tcod*), 106  
 struct\_is\_mandatory() (in module *tcod*), 106  
 suspend() (*tdl.event.App* method), 124  
 sym (*tcod.event.KeyboardEvent* attribute), 44  
 sys\_check\_for\_event() (in module *tcod*), 88  
 sys\_elapsed\_milli() (in module *tcod*), 87  
 sys\_elapsed\_seconds() (in module *tcod*), 87  
 sys\_force\_fullscreen\_resolution() (in module *tcod*), 87  
 sys\_get\_char\_size() (in module *tcod*), 88  
 sys\_get\_current\_resolution() (in module *tcod*), 88  
 sys\_get\_fps() (in module *tcod*), 86  
 sys\_get\_last\_frame\_length() (in module *tcod*), 87  
 sys\_get\_renderer() (in module *tcod*), 87  
 sys\_register\_SDL\_renderer() (in module *tcod*), 88  
 sys\_save\_screenshot() (in module *tcod*), 87  
 sys\_set\_fps() (in module *tcod*), 86  
 sys\_set\_renderer() (in module *tcod*), 87  
 sys\_sleep\_milli() (in module *tcod*), 87  
 sys\_update\_char() (in module *tcod*), 88  
 sys\_wait\_for\_event() (in module *tcod*), 88

## T

*tcod* (module), 1  
*tcod.bsp* (module), 29  
*tcod.COLCTRL\_1* (built-in variable), 74  
*tcod.COLCTRL\_2* (built-in variable), 74  
*tcod.COLCTRL\_3* (built-in variable), 74  
*tcod.COLCTRL\_4* (built-in variable), 74  
*tcod.COLCTRL\_5* (built-in variable), 75  
*tcod.COLCTRL\_BACK\_RGB* (built-in variable), 75  
*tcod.COLCTRL\_FORE\_RGB* (built-in variable), 75  
*tcod.COLCTRL\_STOP* (built-in variable), 75  
*tcod.console* (module), 33  
*tcod.event* (module), 43  
*tcod.EVENT\_ANY* (built-in variable), 86  
*tcod.EVENT\_FINGER* (built-in variable), 86  
*tcod.EVENT\_FINGER\_MOVE* (built-in variable), 86  
*tcod.EVENT\_FINGER\_PRESS* (built-in variable), 86  
*tcod.EVENT\_FINGER\_RELEASE* (built-in variable), 86  
*tcod.EVENT\_KEY* (built-in variable), 86  
*tcod.EVENT\_KEY\_PRESS* (built-in variable), 86  
*tcod.EVENT\_KEY\_RELEASE* (built-in variable), 86  
*tcod.EVENT\_MOUSE* (built-in variable), 86  
*tcod.EVENT\_MOUSE\_MOVE* (built-in variable), 86  
*tcod.EVENT\_MOUSE\_PRESS* (built-in variable), 86  
*tcod.EVENT\_MOUSE\_RELEASE* (built-in variable), 86  
*tcod.EVENT\_NONE* (built-in variable), 86  
*tcod.image* (module), 51  
*tcod.map* (module), 55  
*tcod.noise* (module), 59  
*tcod.path* (module), 63  
*tcod.random* (module), 67  
*tcod.tileset* (module), 69  
*tdl* (module), 109  
*tdl.event* (module), 121  
*tdl.map* (module), 127  
*tdl.noise* (module), 131  
 TDLError, 112  
 text (*tcod.event.TextInput* attribute), 47  
 text (*tcod.Key* attribute), 84  
 TextInput (class in *tcod.event*), 47  
 tile (*tcod.event.MouseButtonEvent* attribute), 46  
 tile (*tcod.event.MouseMotion* attribute), 45  
 tile\_height (*tcod.tileset.Tileset* attribute), 69  
 tile\_motion (*tcod.event.MouseMotion* attribute), 45  
 tile\_shape (*tcod.tileset.Tileset* attribute), 69  
 tile\_width (*tcod.tileset.Tileset* attribute), 69  
 tiles (*tcod.console.Console* attribute), 40  
 tiles2 (*tcod.console.Console* attribute), 40  
 Tileset (class in *tcod.tileset*), 69  
 transparent (*tcod.map.Map* attribute), 55  
 transparent (*tdl.map.Map* attribute), 128  
 type (*tcod.event.Event* attribute), 43  
 type (*tcod.event.KeyboardEvent* attribute), 44  
 type (*tcod.event.MouseButtonEvent* attribute), 46  
 type (*tcod.event.MouseMotion* attribute), 45  
 type (*tcod.event.MouseWheel* attribute), 46  
 type (*tcod.event.Quit* attribute), 44  
 type (*tcod.event.TextInput* attribute), 47  
 type (*tcod.event.WindowEvent* attribute), 47  
 type (*tcod.event.WindowMoved* attribute), 47  
 type (*tcod.event.WindowResized* attribute), 47  
 type (*tdl.event.Event* attribute), 121

## U

Undefined (class in *tcod.event*), 48  
 uniform() (*tcod.random.Random* method), 68  
 update() (*tdl.event.App* method), 124

## V

vflip() (*tcod.image.Image* method), 54  
 vk (*tcod.Key* attribute), 84  
 vline() (*tcod.console.Console* method), 39

## W

wait() (in module *tcod.event*), 50  
 wait() (in module *tdl.event*), 124  
 walk() (*tcod.bsp.BSP* method), 31

walkable (*tcod.map.Map* attribute), 55  
walkable (*tdl.map.Map* attribute), 128  
wheel\_down (*tcod.Mouse* attribute), 86  
wheel\_up (*tcod.Mouse* attribute), 85  
width (*tcod.bsp.BSP* attribute), 29  
width (*tcod.console.Console* attribute), 41  
width (*tcod.event.WindowResized* attribute), 47  
width (*tcod.image.Image* attribute), 51  
width (*tcod.map.Map* attribute), 55  
Window (*class in tdl*), 117  
WindowEvent (*class in tcod.event*), 47  
WindowMoved (*class in tcod.event*), 47  
WindowResized (*class in tcod.event*), 47  
write() (*tdl.Console* method), 117

## X

x (*tcod.bsp.BSP* attribute), 29  
x (*tcod.event.MouseWheel* attribute), 46  
x (*tcod.event.Point* attribute), 43  
x (*tcod.event.WindowMoved* attribute), 47  
x (*tcod.Mouse* attribute), 85

## Y

y (*tcod.bsp.BSP* attribute), 29  
y (*tcod.event.MouseWheel* attribute), 46  
y (*tcod.event.Point* attribute), 43  
y (*tcod.Mouse* attribute), 85