Workshop Python

Text User Interfaces (TUIs) mit curses

1. **Allgemeines zu curses**
2. **Basics zu curses mit Python**
3. **Funktionen der curses library**
4. **Konstanten der curses library**

- Ganz am Anfang: Telegraphie als das "Viktorianische Internet"
- Morsecode als erstes "Online-Protokoll", Morsestationen als die ersten "Proto"-Terminals.
- Weiterentwicklung der Morse-Telegraphie: **Fernschreiber**
  (Engl.: *TeleTYpeWriter* -> `tty`)
- Fernschreiber: Einsatz als frühe Computer-Terminals, aber nur suboptimal für Interaktion mit IT:
  - Papierverbrauch, Ausdruck meistens unnötig
  - Zu langsam
  - Keine komplexere Interaktion (editieren etc.)

- Ab den 1970ern: Aufkommen der Computerterminals

Die curses library

- Entwickelt von Ken Arnold ab 1980 für BSD Unix
- Aufbauend auf Code aus **VI** (1976)
- Name *curses* Verballhornung von *'cursor optimization'*
- Eine der ersten Anwendungen für curses: **Rogue** (ebenfalls von Ken Arnold)

Weiterentwicklungen:

- pcurses (ab 1982, u.a. Unterstützung von Farben und Attributen)
- ncurses (**new** curses, ab 1993),
  ist heute *die* Bibliothek die als ***curses*** bezeichnet wird
  - Eines der wenigen GNU-Projekte, die **nicht** unter der GPL lizensiert sind (eigene, MIT-ähnliche Lizenz)

Warum eigentlich noch Text Interfaces, wenn es doch GUIs gibt?

- ncurses ist eher verfügbar als GUI-Libraries
- Erheblich geringer Ressourcenverbrauch
- Weniger Probleme mit Videotreibern

- Embedded systems
- Systemprogramme (Installationsoberflächen, Administration etc.)
- Rapid Prototyping

  - `cfdisk`
  - `make menuconfig`
  - `aptitude`
  - `Midnight Commander`
  - `iptraf`
  - `powertop`

Die Python-Library **`curses`** ist prinzipiell ein *Wrapper* um **`libncurses`** mit einigen nützlichen Zusatzfunktionen.

Achtung: Manche Features von ncurses sind (noch) nicht implementiert!

`curses`	**(OK)**	Basispaket zu ca. 80% implementiert
`curses.panel`	**OK**	Überlappende Fenster
`curses.menu`	**X**	Menü-Strukturen (fehlt)
`curses.forms`	**X**	Eingabeformulare (fehlt)

Python-spezifische Erweiterungen von Curses:

`curses.ascii`	**OK**	ASCII-Utilities
`curses.textpad`	**OK**	Einfaches Texteingabefeld
`PyCDK`	**(OK)**	Alpha, seit 2009 nicht mehr weiterentwickelt

Initialisierung eines Curses-Programms:

```
import curses

stdscr = curses.initscr() # Initialisiere den Screen
curses.noecho()           # Deaktiviere Tastaturecho
curses.cbreak()           # Deaktiviere Line-Buffering

# Optional, aber eigentlich immer verwendet:

stdscr.keypad(1)          # Aktiviere ESC-Sequenzen
curses.start_color()      # Aktiviere Farben
```

Finalisierung eines Curses-Programms:

```
stdscr.keypad(0)            # Deaktiviere ESC-Sequenzen

curses.nocbreak()           # Reaktiviere Linebuffering
curses.echo()               # Reaktiviere Tastaturecho
curses.endwin()             # Schliesse curses-Fenster
```

Sollte unbedingt ausgeführt werden, da sonst das Terminal in unbrauchbarem Zustand zurückbleibt!

-> Hauptprogramm mit `try / finally` Exceptionhandler

```
import curses

def initscreen():
    stdscr = curses.initscr()
    curses.noecho(); curses.cbreak(); stdscr.keypad(1)
    curses.start_color()
    return stdscr

def restorescreen():
    stdscr.keypad(0); curses.nocbreak(); curses.echo()
    curses.endwin()

def main(stdscr):
    <Program>

if __name__ == "__main__":
    try:
        main(initscreen())
    finally:
        restorescreen()
```

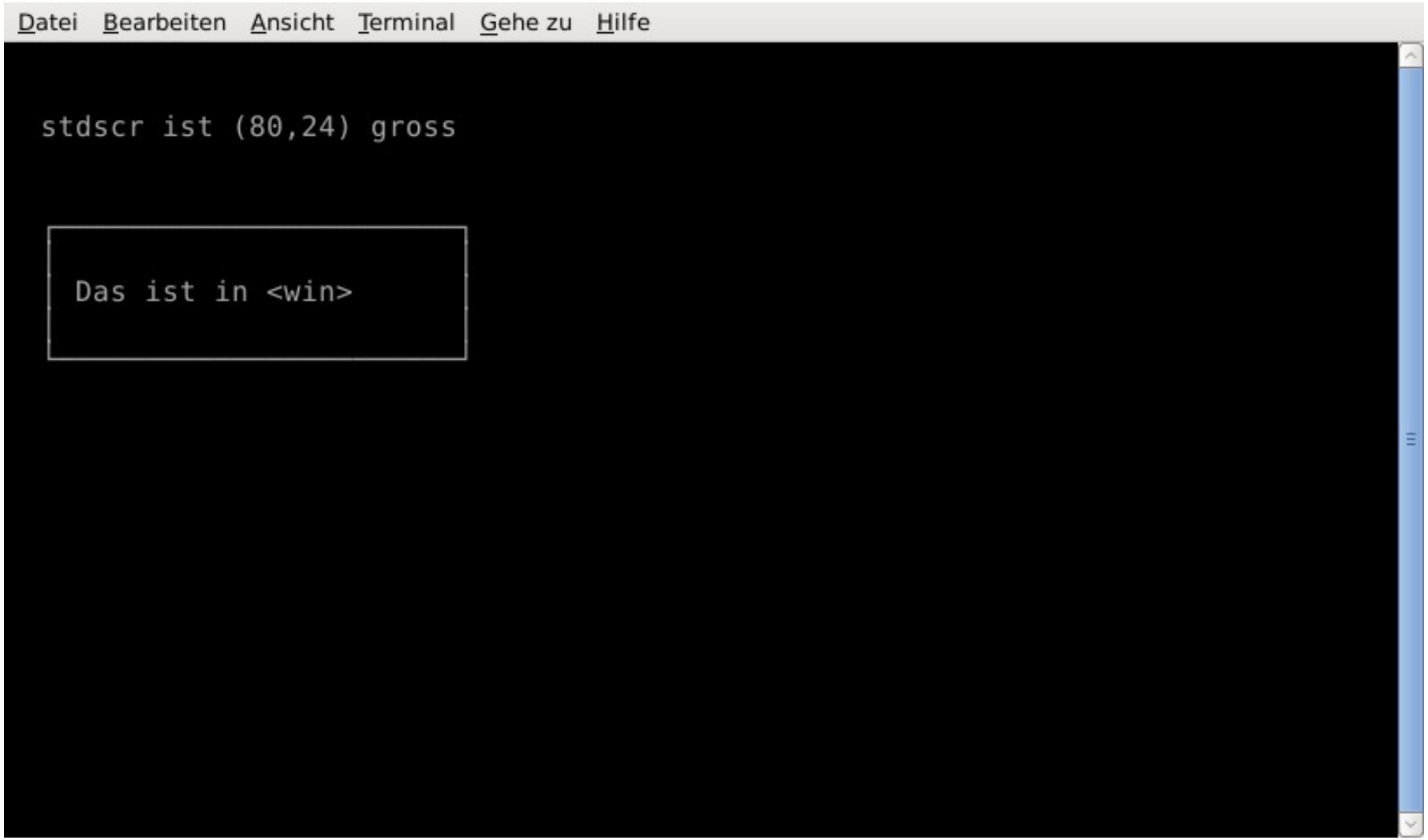
```
import curses

def main(stdscr):
    <Program>

if __name__ == "__main__":
    curses.wrapper(main)
```

Standard Initialisierung / Finalisierung wird von `curses.wrapper` übernommen

```
import curses

def main(stdscr):
   <Program>

if __name__ == "__main__":
   curses.wrapper(main)
```

```
import curses

def main(stdscr, *args, **kwargs):
   <Program>

if __name__ == "__main__":
   curses.wrapper(main, *args, **kwargs)
```

Standard Initialisierung / Finalisierung wird von `curses.wrapper` übernommen


Datei Bearbeiten Ansicht Terminal Gehe zu Hilfe
stdscr ist (80,24) gross
Das ist in <win>

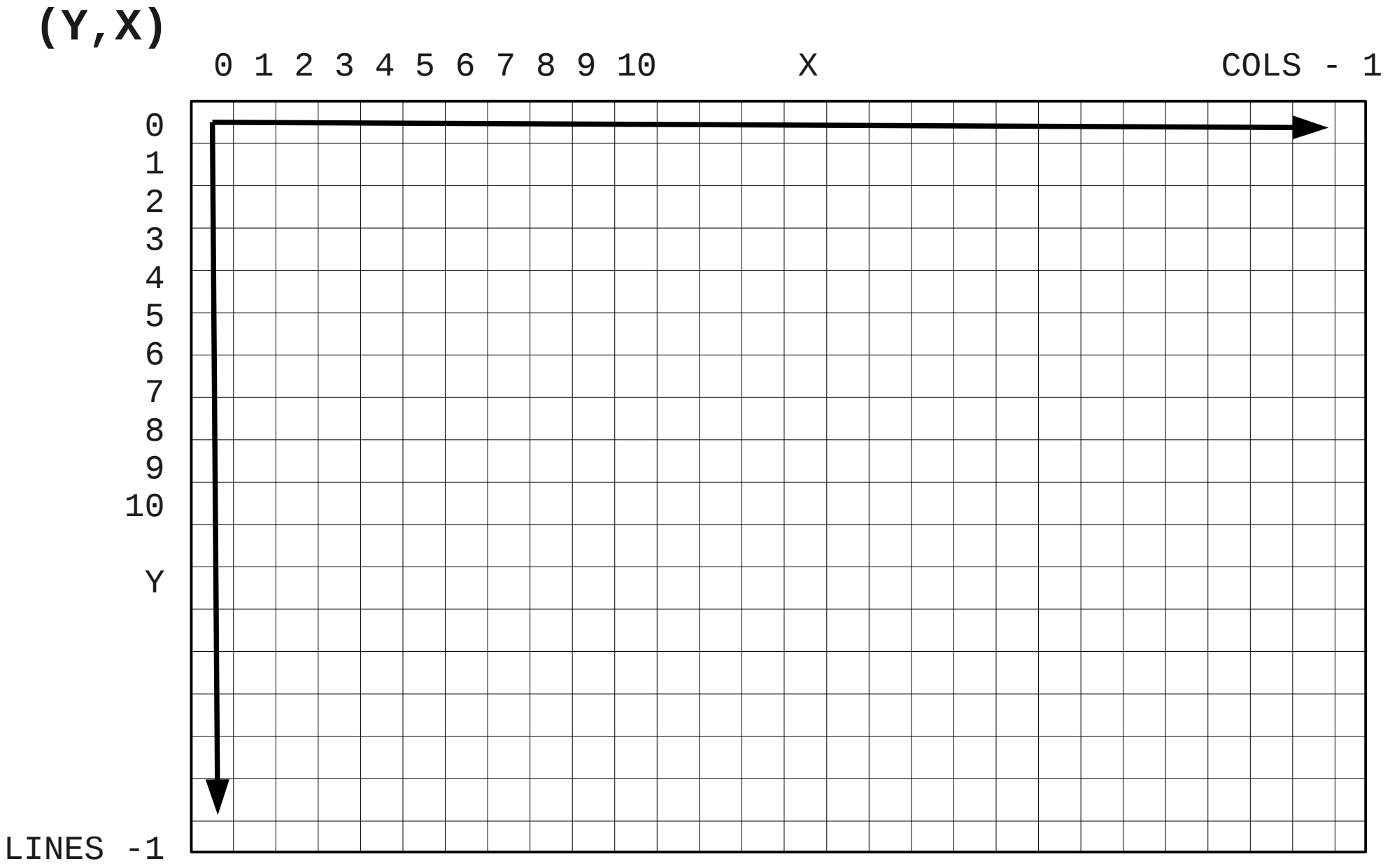
```
import curses

def main(stdscr):
    curses.curs_set(False)

    h, w = stdscr.getmaxyx()
    stdscr.addstr(2,2,"stdscr ist ("+str(w)+","+str(h)+") gross")
    stdscr.refresh()

    win = curses.newwin(5,25,5,2)
    win.border()
    win.addstr(2,2,"Das ist in <win>")
    win.refresh()

    while True:
        c = stdscr.getch()
        if c == ord('q'): break
    return

if __name__ == "__main__":
    curses.wrapper(main)
```

Die curses-Library ist in zwei Gruppen aufgeteilt:

- Methoden, die sich auf die **curses**-Umgebung beziehen
  - Initialisierung, Finalisierung, Erzeugung von Fenstern, ...
  - Enthält auch die curses-Konstanten (Farbnamen, Sonderzeichen etc.)
  - Aufruf per **`curses.`**`<method>` / **`curses.`**`<constant>`

- Methoden, die Fenster (Windows) beeinflussen
  - Aufruf per **`<window>.`**`<method>`
  - Der Hauptscreen `stdscr` ist auch ein Window!

```
import curses

def main(stdscr):
    curses.curs_set(False)

    h, w = stdscr.getmaxyx()
    stdscr.addstr(2,2,"stdscr ist (" + \
                      str(w) + "," + str(h) + ") gross")
    stdscr.refresh()

    win = curses.newwin(5,25,5,2) # Height, Width, y, x
    win.border()
    win.addstr(2,2,"Das ist in <win>")
    win.refresh()

    stdscr.getch()
    return

if __name__ == "__main__":
    curses.wrapper(main)
```


(Y,X)
0 1 2 3 4 5 6 7 8 9 10
X
COLS - 1
0
1
2
3
4
5
6
7
8
9
10
Y
LINES -1

Aufgepasst: Änderungen an Fenstern werden nicht automatisch angezeigt.

Nachdem der Inhalt eines Fensters modifiziert wurde, muss der Screen neu aufgefrischt werden, erst dann werden Änderungen sichtbar:

```
<window>.refresh()          # Update <window> immediately
```

Falls mehrere Fenster modifiziert wurden, ist es performanter (und reduziert Bildschirmflackern), zuerst die Fenster als geändert zu kennzeichnen, und danach den Screen refresh sichtbar zu machen:

```
<window1>.noutrefresh()     # Mark <window1> for refresh
<window2>.noutrefresh()     # Mark <window2> for refresh
...
curses.doupdate()           # Display all updates
```

`refresh()` ist nichts anderes als `noutfresh()` sofort gefolgt von `doupdate()`

- Example 01: Grundlegende Struktur eines Python curses Programms
- Example 02: Output von Strings und chtype Characters
- Example 03: Farben und Attribute
- Example 04: Windows und Subwindows
- Example 05: Cursor
- Example 06: Input
- Example 07: String input
- Example 08: Timeout-driven input
- Example 09: Scrolling
- Example 10: Pads (virtuelle Screens)
- Example 11: Panels (überlappende Windows)
- Example 12: Mausevents
- Example 13: UTF-8

```
curses.wrapper(<mainfunc> [, *args, **kwargs])
```

Grundlegende Initialisierung / Finalisierung der Curses-Oberfläche

Ruft <mainfunc> mit Argumenten **`(stdscr, *args, **kwargs)`** auf.

```
<window>.refresh()
```

Aktualisiert sofort den Inhalt von <window> auf dem Bildschirm

```
<window>.noutrefresh()
curses.doupdate()
```

(noutrefresh): Markiert <window> für Aktualisierung
(doupdate): Führt alle markierten Aktualisierungen auf dem Bildschirm durch

```
curses.newwin([height, width,] starty, startx)
```

Erzeugt ein neues Fenster an (starty, startx) mit Breite / Höhe (width / height)

```
<window>.derwin([height, width,] starty, startx)
```

Erzeugt ein neues Subfenster an (starty, startx) (relativ zu <window>) mit Breite / Höhe (width / height)

```
(y, x) = <window>.getmaxyx()     # Window size
(y, x) = <window>.getbegyx()     # Window origin coordinates
(y, x) = <window>.getparyx()     # Window origin rel. parent
```

Holt die Größe (getmaxyx), Ursprungskoordinaten (getbegyx), oder Koordinaten relativ zum Elternfenster (getparyx)

```
<bool> = <window>.enclose(y, x)
```

Test ob die Koordinaten (y, x) innerhalb von <window> liegen.

```
<window>.addstr([y, x,] string [, attr])
<window>.addnstr([y, x,] string, num [, attr])
```

Gibt einen String (8-bit ASCII) ab der aktuellen Cursorposition / ab (y,x) aus

```
<window>.addch([y, x,] char [, attr])
```

Gibt ein Zeichen (chtype) ab dem aktuellen Cursor / ab (y,x) aus

```
<window>.hline([y, x,] char, num)
<window>.vline([y, x,] char, num)
```

Zeichnet eine horizontale / vertikale Linie der Länge <num> mit <char>

```
<window>.border()
```

Zeichnet einen Rahmen um <window>

```
<window>.scrollok(<bool>)
```

Schaltet das Scrolling-Verhalten von <window> ein / aus.

```
<window>.scroll([lines=1])
```

Scrollt den Inhalt von <window> um lines Zeilen (negativ: nach unten scrollen)

```
<window>.insch([y, x,] ch [,attr])
```

Wie **addch**, rückt aber alle nachfolgenden Zeichen um eins nach hinten

```
<window>.insstr([y, x,] string [,attr])
<window>.insnstr([y, x,] string, num [,attr])
```

Wie **addstr / addnstr**, rückt aber alle nachfolgenden Zeichen nach hinten

```
<window>.insertln()
```

Fügt an Cursorposition eine leere Zeile ein; rückt nachfolgende Zeilen nach unten

```
<window>.clrtoeol()
```

Löscht den Inhalt von <window> von der Cursorposition bis zum Ende der Zeile.

```
<window>.clrtobot()
```

Löscht den Inhalt von <window> von der Cursorposition bis zum Ende des Fensters.

```
<window>.clear()
```

Löscht den kompletten Inhalt von <window>

```
<window>.delch([y, x])
```

Lösche Char an Cursorposition / an (y,x), rücke nachfolgende Zeichen eins nach vorne

```
<window>.deleteln()
```

Löscht die Zeile an der Cursorposition; rücke nachfolgende Zeilen eins nach oben

```
curses.curs_set(state)
```

Setzt die Sichtbarkeit des Cursors:

- `0` / `False`: Unsichtbar
- `1` / `True`: Sichtbar
- 2: Maximale Sichtbarkeit (abhängig vom Terminal)

```
<window>.move(newy, newx)
```

Setzt die Position des Cursors in <window> auf (newy / newx)

```
y, x = <window>.getyx()
```

Holt die aktuelle Position des Cursors in <window>

```
c = <window>.getch()
c = <window>.getstr([num])
```

`getch`: Holt ein chtype Zeichen
`getstr`: Holt einen String (max. num Zeichen)

von der Tastatureingabe assoziiert mit <window>

```
c = <window>.inch([y, x])
st = <window>.instr([y, x] [, num])
```

`inch`: Holt ein chtype Zeichen
`instr`: Holt einen String (max. n Zeichen)

aus dem Inhalt von <window> von der aktuellen Cursurposition / von (y, x)

```
<window>.timeout(time)
```

Setzt das Blocking / Non-blocking Verhalten von getch():

- < 0: getch wartet unendlich lange auf Eingabe
- 0: getch gibt kontinuierlich -1 / curses.ERR zurück falls keine Eingabe vorliegt
- > 0: getch wartet time Millisekunden bevor curses.ERR zurückgegeben wird

```
curses.echo()
curses.noecho()
```

Schaltet echo-Modus ein / aus (Tastatureingaben werden am Bildschirm angezeigt)

```
curses.ungetch(ch)
```

Fügt <ch> in den Eingabepuffer ein; nächster Aufruf von getch() liefert <ch> (Achtung: Maximal 1 Zeichen im Eingabepuffer!)

```
<window>.attrset(attr)
```

Setzt die aktuellen Zeichenattribute / Farben auf <attr>

```
<window>.attron(attr)
<window>.attroff(attr)
```

Schaltet das Zeichenattribut / Farbenpaar <attr> ein / aus

```
<window>.chgat([y, x,] [num,] attr)
```

Ändert die Attribute / Farben der Zeichen in <window> auf <attr>
(ab Cursorposition / an (y, x), maximal <num> Zeichen)

```
<window>.bkgd([ch, attr])
```

Ändert die Attribute / Farben aller Zeichen in <window> auf <attr>
(Optional: Füllt den Hintergrund mit dem Zeichen <ch>)

```
<bool> = curses.has_colors()
```

Test, ob das Terminal Farben darstellen kann

```
<attrs> = curses.termattrs()
```

Holt alle darstellbaren Attribute (als logisches OR)

```
curses.init_pair(pairnum, foreground, background)
```

Weist <pairnum> eine Kombination Vordergrund / Hintergrundfarbe zu

```
<color> = curses.color_pair(pairnum)
```

Holt das Farbpaar <pairnum> (kann mit anderen Attributen über logisches OR kombiniert werden)

```
curses.beep()
```

Akustischer "Beep"

```
curses.flash()
```

Visueller "Flash" (setzt den ganzen Bildschirm kurz Reverse)

```
curses.napms(<time>)
```

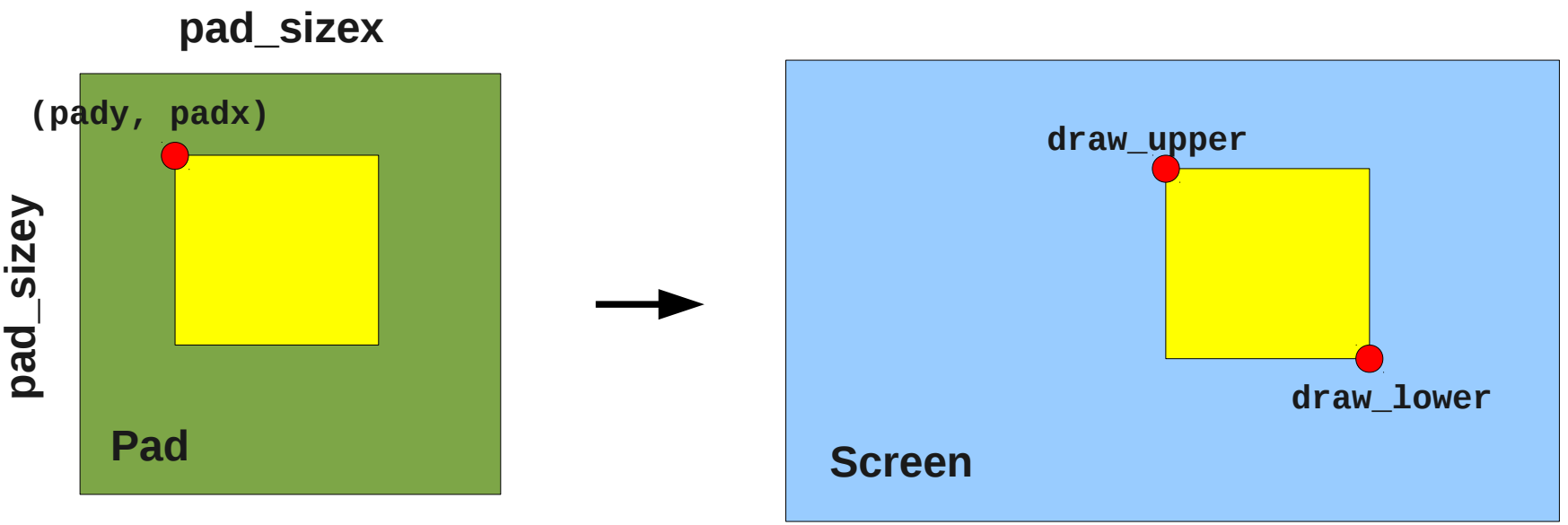
Wartet <time> Millisekunden

```
<pad> = curses.newpad(pad_sizey, pad_sizex)
```

Erzeugt ein neues Pad der Größe (sizey, sizex)

```
<pad>.noutrefresh(pady, padx,
                  pdrawy_upper, pdrawx_upper,
                  pdrawy_lower, pdrawx_lower)
<pad>.refresh(<...>)
```

Aktualisiert den Inhalt des Pads:



```
<panel> = curses.panel.new_panel(<window>)
```

Erzeugt ein neues Panel assoziiert mit <window>

```
curses.panel.update_panels()
```

Erneuert den virtuellen Screen nach Änderungen im Panel Stack
(danach muss noch curses.doupdate() aufgerufen werden)

```
<panel>.show()
<panel>.hide()
```

Zeigt an / versteckt das Panel <panel>

```
<panel>.top()
<panel>.bottom()
```

Verschiebt das Panel <panel> ganz nach vorne / ganz nach hinten

```
(availmask, oldmask) = curses.mousemask(<newmask>)
```

Setzt die abzufangenden Mausereignisse <newmask>.
Liefert die vorherige Mausmaske (oldmask)
und die verfügbaren angeforderten Ereignisse (availmask)

```
(id, x, y, z, bstate) = curses.getmouse()
```

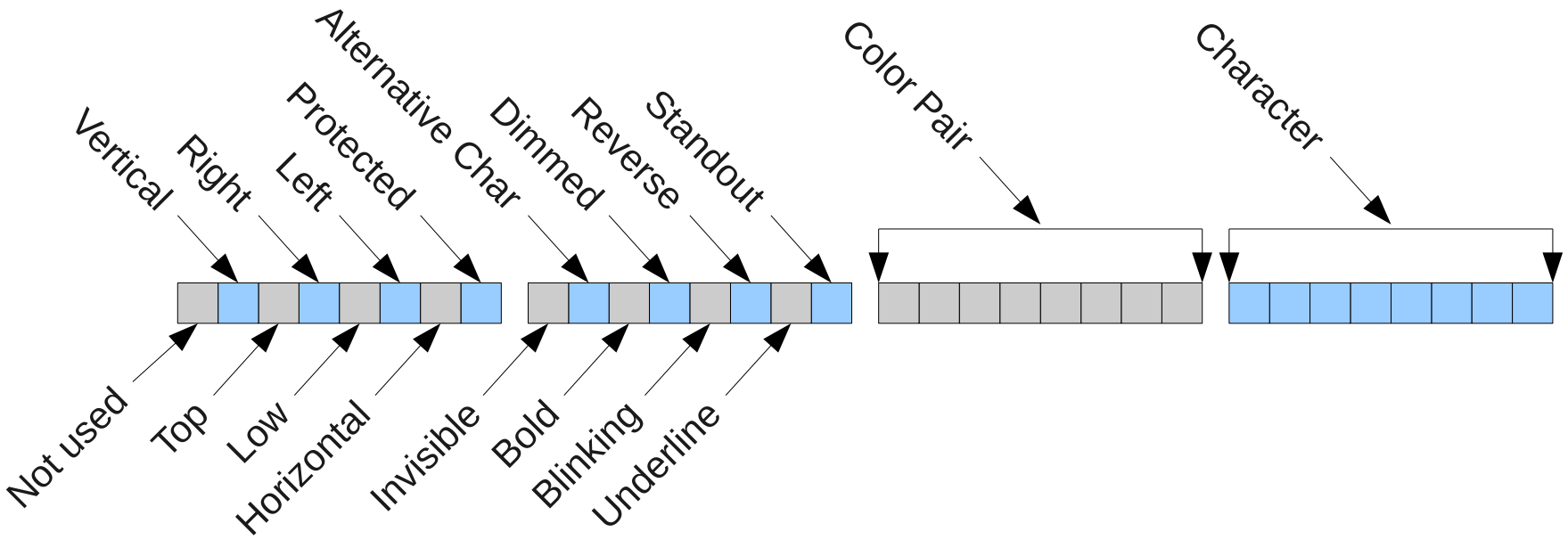
Holt das aktuelle Mausereignis (nach getch() == KEY_MOUSE)

- id: ID der jeweiligen Maus
- x, y, z: Koordinaten des Ereignisses (z derzeit ohne Funktion)
- bstate: Aufgetretenes Mausereignis

```
curses.mouseinterval(<time>)
```

Setzt das Zeitintervall zur Erkennung von Klicks auf <time> Millisekunden
(Default: 200 ms)

Das **`chtype`** Format:



```
A_NORMAL        Normal display (no highlight)
A_BOLD          Extra bright or bold
A_UNDERLINE     Underlining
A_REVERSE       Reverse video

COLOR_PAIR(n) Color-pair number n

A_ALTCHARSET  Alternate character set
A_CHARTEXT    Bit-mask to extract a character

A_STANDOUT    Best highlighting mode of the terminal.
A_BLINK       Blinking
A_DIM         Half bright
A_PROTECT     Protected mode
A_INVIS       Invisible or blank mode
```

COLOR_BLACK

COLOR_WHITE

COLOR_RED

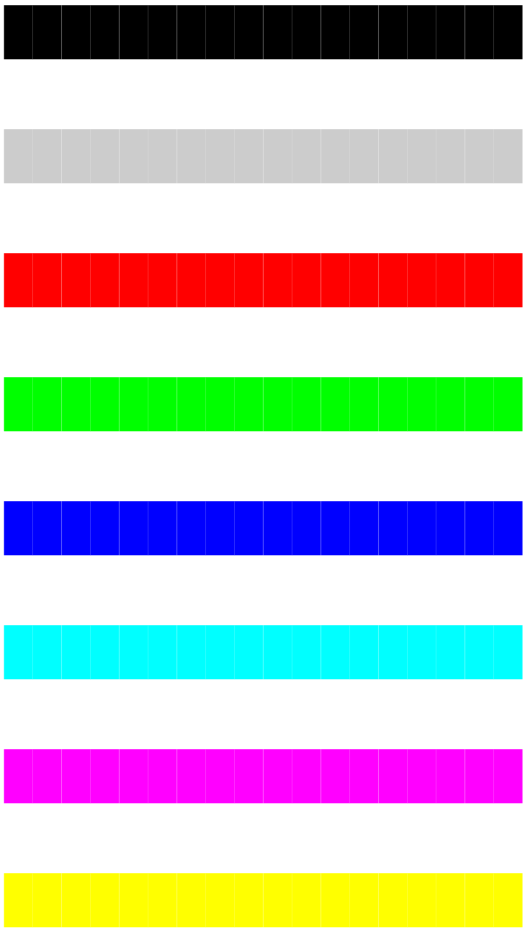
COLOR_GREEN

COLOR_BLUE

COLOR_CYAN

COLOR_MAGENTA

COLOR_YELLOW



```
Linien/Rahmenelemente:

    ┌ ACS_ULCORNER      ├ ACS_LTEE        ─ ACS_HLINE
    ┐ ACS_URCORNER      ┤ ACS_RTEE        │ ACS_VLINE
    └ ACS_LLCORNER      ┴ ACS_BTEE        ┼ ACS_PLUS
    ┘ ACS_LRCORNER      ┬ ACS_TTEE

Blöcke:

    ▒ ACS_CKBOARD       0x20              0x20|curses.A_REVERSE
                                          ACS_BLOCK

Scan Lines:

    ⎺ ACS_S1      ⎻ ACS_S3      ─ ACS_S7      _ ACS_S9

Diverse Symbole:

    ◆ ACS_DIAMOND       · ACS_BULLET      ± ACS_PLMINUS
```

```
KEY_UP        Cursor Up
KEY_DOWN      Cursor Down
KEY_LEFT      Cursor Left
KEY_RIGHT     Cursor Right

KEY_F1        F1
KEY_F2        F2
KEY_F3        F3
KEY_F4        F4

KEY_IC        Insert
KEY_DC        Delete
KEY_HOME      Home / Pos1
KEY_END       End
KEY_PPAGE     Page Up
KEY_NPAGE     Page Down

KEY_RSIZE     Window resize event
KEY_MOUSE     Mouse event
ERR           Error / No input
```

```
<n> = 1, 2, 3, 4

BUTTON<n>_PRESSED              Maustaste gedrückt
BUTTON<n>_RELEASED             Maustaste losgelassen
BUTTON<n>_CLICKED              Maustaste Einfachklick
BUTTON<n>_DOUBLE_CLICKED       Maustaste Doppelklick
BUTTON<n>_TRIPLE_CLICKED       Maustaste Dreifachklick

BUTTON_SHIFT                   Shift war gedrückt
BUTTON_CTRL                    Ctrl war gedrückt
BUTTON_ALT                     Alt war gedrückt

ALL_MOUSE_EVENTS               Alle möglichen Mouseevents
```