semesterplanner-lua — Semesterplanner package in lua with tikz only*

Lukas Heindl
: https://gitlab.com/AtticusSullivan/semesterplanner-lua

Released ?

Abstract

This package provides a mean to easily print a timetable e.g. for a semesterplan. The reason for this package to exist is that I wanted to reimplement https://github.com/nlschn/semesterplanner/ with printing the timetable with `tikz` only (which is more easily to be modified) and with the ability to make entries spanning only a fraction of the column (for showing simultanious events).

Documents using this package need to be compiled with LuaLaTeX. The package requires `xcolor`, `fontawesome`, `tikz` (and `pgfkeys`).

Contents

1 Usage **1**
1.1 timetable . . . 1
1.1.1 Special Notes . . . 2
1.1.2 Example . . . 3
1.2 Icons . . . 3

2 Implementation **4**
2.1 semesterplanner-lua.sty . . . 4
2.1.1 Global Stuff . . . 4
2.1.2 Local Stuff (timetable-env local) . . . 4
2.2 semesterplanner-lua.lua . . . 8

3 Change History **11**

4 Index **11**

1 Usage

1.1 timetable

`timetable` `\begin{timetable}[opts]\ldots\end{timetable}`
`opts` are of course optional arguments:

`days` List of the names of the days that should be set as column names. Note that if you specify only 4 names only these 4 columns will be printed (with the first day being identified as Monday) *Default:* `Mon,Thue,Wend,Thur,Fri`

`start time` Explicit start-time of the timetable given in minutes (`HH*60 + MM`). Can be set as `start time/.evaluated={HH*60 + MM}`. If this is empty, the start time is derived from the given events. *Default:* `""`

`end time` Equivalent to `start-time` *Default:* `""`

*This file describes version ?, last revised ?.

`width` Give the width of the timetable. (can be given e.g. as `\textwidth` as this is directly given to tikz). *Default:* `\textwidth`

`length` Give the length of the timetable (measured in `cm`) (has to be a straight number since this is needed in calculation) *Default: 10*

This is the core environment of this package. Within it you can use `\lecture`, `\seminar`, `\tutorial`, `\officehour` and `\meeting`. All these commands are only defined inside the `timetable` environment, and have the same structure.

```
\lecture       \lecture   {Name}{Lecturer}{Place}{Day}{Time}{Priority}{Event-code}
\tutorial      \tutorial  {Name}{Lecturer}{Place}{Day}{Time}{Priority}{Event-code}
\seminar       \seminar   {Name}{Lecturer}{Place}{Day}{Time}{Priority}{Event-code}
\officehour    \officehour{Name}{Lecturer}{Place}{Day}{Time}{Priority}{Event-code}
\meeting       \meeting   {Name}{Lecturer}{Place}{Day}{Time}{Priority}{Event-code}
```

`Name` Give the name of the lecture

`Lecturer` Give the name of the lecturer

`Place` Give the place of the event (most probably the room or an online plattform, see 1.2)

`Day` The weekday on which the event takes place. Has to be one of `M`, `T`, `W`, `Th`, `F` for **M**onday, **T**huesday, **W**ednesday, **Th**ursday, **F**riday. Might become customizable in a future version.

`Time` The timespan of the event formatted as `HH:MM-HH:MM` (24H clock)

`Priority` The priority of the event (see 1.2)

`Event-code` Free customizable event code. See the documentation at the end for keys that can be used here (all keys in `/event`). To simply pass arguments to the tikz-node that is being created for the event use `tikz/.append={your arguments}` (be careful with `text width`, `text height`, `text depth` as these keys are being used for the dimensions of the node as well as with `anchor`)

The entries `Day` and `Time` are mandatory since they are needed for the positioning of the node. All others are merely necessary for the content of the node and are therefore nor mandatory.

1.1.1 Special Notes

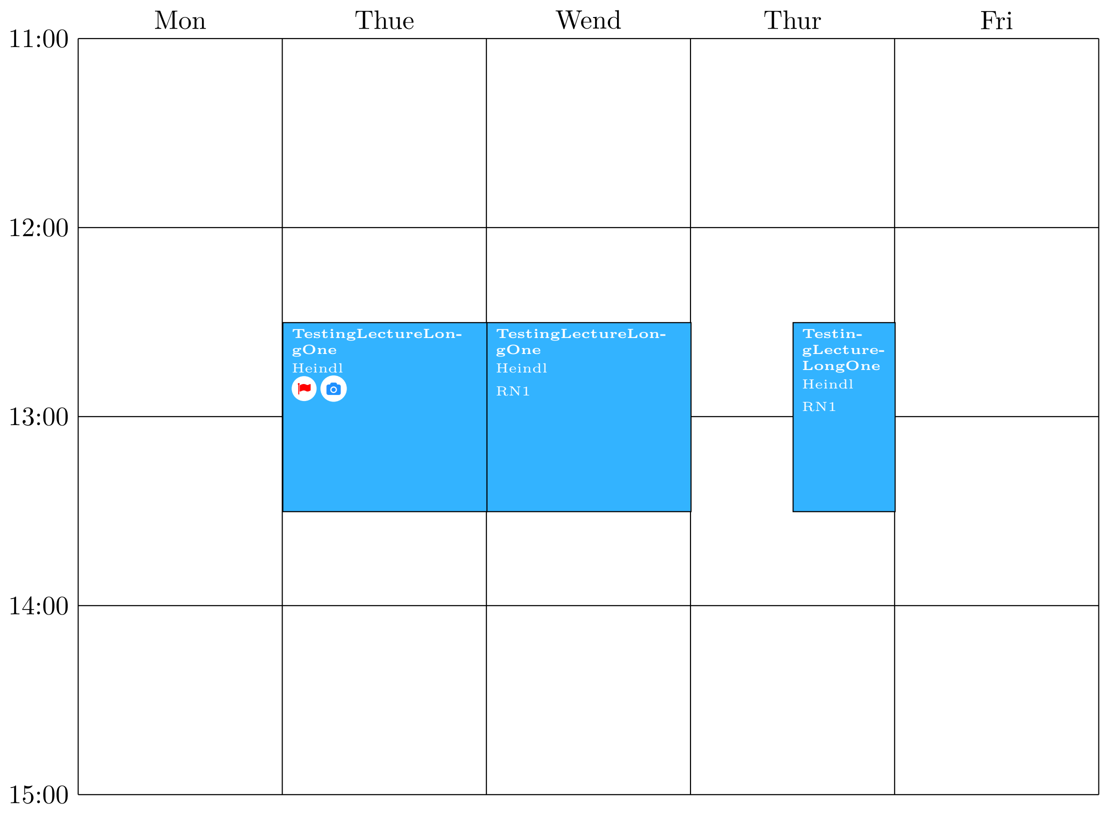
Note that the `length` argument does specify the length of the timetable without taking account of the column headers.

Same goes for the `width` parameter regarding the labels containing the time on the right. Since in this case any tex-lenght is allowed, you can simply try to subtract the length of the clock label using something like `\settowidth{\length}{12:30}` to set a length to the length of a clock label and then subtract this from the length you want to specify.

Hint: The content of the environment isn't processed by this package. Only the event commands (so to speak `\lecture`,`\tutorial`,`\seminar`,`\officehour`,`\meeting` are relevant. All other contents are set immediately before the timetable. Therefore, if you wan to add e.g. a `\hspace*{10cm}` to shift the timetable to the left, the last line of the env would be the place to do so (there musn't be an empty line below since otherwise a new paragraph is started).

1.1.2 Example

```
\begin{timetable}[
        days={Mon,Thue,Wend,Thur,Fri}, start
        time/.evaluated={11*60}, end time/.evaluated={15*60}
    ]
    \lecture{TestingLectureLongOne}{Heindl}{RN1}{W}{12:30-13:30}{}{}
    \lecture{TestingLectureLongOne}{Heindl}{RN1}{Th}{12:30-13:30}{}
         {offset=0.5,scale width=0.5}
    \lecture{TestingLectureLongOne}{Heindl}{\zoom}{T}{12:30-13:30}{\phigh}{}
\end{timetable}
```



1.2 Icons

This package defines some modified fontawesome icons (they are being encircled with a white circle for better readability).

\zoom	📷	\teams	⊞
\BBB	B	\youtube	▶
\pmandatory	⚠	\phigh	⚑
\pmid	⚑	\plow	⚑
\pnone	⊗		
\tbd	?	\tba	📢

2 Implementation

This package uses `semesterplanner-lua` as prefix/directory where possible. Since this is not possible for latex macro names, in this occasions `semesterplannerLua@` is used as prefix.

2.1 semesterplanner-lua.sty

2.1.1 Global Stuff

```
⟨*package⟩
```

Define some colors for the course types (can be globally overwritten)

```
\definecolor{seminar}{rgb}{1.0, 0.8, 0.0}
\definecolor{lecture}{rgb}{0.2, 0.7, 1.0}
\definecolor{tutorial}{rgb}{0.0, 0.8, 0.0}
\definecolor{meeting}{rgb}{0.8, 0.0, 0.0}
\definecolor{officehour}{rgb}{0.0, 0.4, 0.6}
\definecolor{DodgerBlue}{HTML}{1E90FF}
```

Load the lua module

```
\directlua{sp = require("semesterplanner-lua.lua")}
```

2.1.2 Local Stuff (timetable-env local)

`timetable` This is the environment doing all the stuff. To gate the positions where the corresponding macros can be used (and in terms of pgfkeys for reasons of default values) all the macros used are put into the environment.

```
\newenvironment{timetable}[1][]{
```

`\semesterplannerLua@encircle` This macro puts a circle arround its argument for better readability. In this package this is used for the fontawesome symbols.

```
    \newcommand*{\semesterplannerLua@encircle}[1]{
        \begin{minipage}[b][1em][c]{1.5em}
            \begin{tikzpicture}
                \node[fill,circle,inner sep=1pt, color = white] {##1};
            \end{tikzpicture}
        \end{minipage}
    }
```

Set all the pgfkeys required for the arguments. To achieve that the defaults are restored every time the environment is used, this is inside the environment definition. This of course disables all possibilities of setting a global default but enables setting local defaults for the events

```
    \pgfkeys{
```

`/semesterplanner-lua` will be the pgf-path used for this package Set the environment arguments arguments. `days`, `width` and `height` are used later in drawing. `start time` and `end time` are important for collecting the events as well.

`days` is a list of strings representing the header names for the day columns in the timetable (adding Sat and Sun (additional entries) will result in two more columns.

`length` is the vertical length of the timetable (not including the clock labels on the side) measured in cm (in future versions this may become measured in pts for better interaction with the LaTeX lengths.

`width` is the horizontal width of the timetable (not including the column headers on the top) this can be a latex length string or `\textwidth` as well.

`start time` can be used to set a fixed time where the timetable starts (otherwise this is calculated from the entries) to enable this behaviour this key has to be set to `HH*60 + MM` (easy way is by using `start time/.evaluated={HH*60+MM}`)

`end time` equivalent to `start time`

```
        /semesterplanner-lua/.cd,
        days/.initial={Mon,Thue,Wend,Thur,Fri},
        days/.default={Mon,Thue,Wend,Thur,Fri},
        %
        start time/.initial=,
        start time/.default=,
        end time/.initial=,
        end time/.default=,
        %
        width/.initial=\textwidth,
        width/.default=\textwidth,
        length/.initial=10,
        length/.default=10,
        %
```

`/semesterplanner-lua/event` is the path where the keys relevant for the event macro resides

`content` is the content of the event (is passed on without any formatting). Since this is passed to lua without modification its value must be an unexpanded string (lua will simply print it so the eventually the string will be evaluated)

`time` is a `HH:MM-HH:MM` string representing start- and end-time of the event

`day` is either `M`,`T`,`W`,`Th` or `F` specifying the day on which the event takes place

`tikz` this key allows the user to manually pass options to the node created for this event

`scale width` allows to scale the width of the event to be able to draw overlapping events besides each other. Will usually be a value between `0` and `1`.

`offset` same goal like `scale width` but shifts the event node by the given value to the right. (Given as value between `0` and `1` indicating how many columns the event should be shifted)

```
        event/.cd,
        % event arguments
        content/.initial=,
        content/.default=,
        %
        time/.initial=,
        time/.default=,
        day/.initial=,
        day/.default=,
        %
        tikz/.initial=,
        tikz/.default=,
        scale width/.initial=1,
        scale width/.default=1,
        offset/.initial=0,
        offset/.default=0,
    }
```

Commands for symbols of priority

`\pmandatory`

```
    \protected\def\pmandatory{\semesterplannerLua@encircle{\textcolor{red}{\faWarning}}}
```

`\phigh`

```
    \protected\def\phigh{\semesterplannerLua@encircle{\textcolor{red}{\faFlag}}}
```

`\pmid`

```
    \protected\def\pmid{\semesterplannerLua@encircle{\textcolor{yellow}{\faFlag}}}
```

`\plow`

```
    \protected\def\plow{\semesterplannerLua@encircle{\textcolor{green}{\faFlag}}}
```

`\pnone`

```
\protected\def\pnone{\semesterplannerLua@encircle{\textcolor{gray}{\faTimesCircle}}}
```

Commands for online platforms.

`\teams`

```
\protected\def\teams{\semesterplannerLua@encircle{\textcolor{DodgerBlue}{\faWindows}}}
```

`\zoom`

```
\protected\def\zoom{\semesterplannerLua@encircle{\textcolor{DodgerBlue}{\faCamera}}}
```

`\youtube`

```
\protected\def\youtube{\semesterplannerLua@encircle{\textcolor{red}{\faYoutubePlay}}}
```

`\BBB`

```
\protected\def\pmandatory{\semesterplannerLua@encircle{\textcolor{red}{\faBold}}}
```

Command for ”To be determined” and ”To be Announced”

`\tbd`

```
\protected\def\tbd{\faQuestion}
```

`\tba`

```
\protected\def\tba{\faBullhorn}
```

Read the argumens given by the user after restoring the defaults (Restoring currently makes no sense, since they are created a few lines above anyways, but creation might be moved outside the environment some day.
Afterwards the lua module is beeing initialized (erase data from possible previous runs.

```
\pgfkeys{/semesterplanner-lua/.cd, days,length,width,start time,end time, #1}
\directlua{sp.init(
    "\pgfkeysvalueof{/semesterplanner-lua/days}",
    "\pgfkeysvalueof{/semesterplanner-lua/start time}",
    "\pgfkeysvalueof{/semesterplanner-lua/end time}")}
```

`\semesterplanner@event` Is used to pass the event to the lua engine which in turn will collect the event to draw it in the end. For that the arguments given are parsed after restoring the pgf keys to their default values. The optional argument herby is a sequence of pgf keys, the second argument is a string representing the content (this MUST be unexpanded since this is passed to lua which in turn will pass it unmodified back)

```
\newcommand{\semesterplannerLua@event}[2][]{
    \pgfkeys{/semesterplanner-lua/event/.cd,content,time,day,tikz,scale width,
    offset, ##1, content=##2}
    \directlua{
        sp.addEvent{
            time="\pgfkeysvalueof{/semesterplanner-lua/event/time}",
            day="\pgfkeysvalueof{/semesterplanner-lua/event/day}",
            tikz=[[\pgfkeysvalueof{/semesterplanner-lua/event/tikz}]],
            content=[[\pgfkeysvalueof{/semesterplanner-lua/event/content}]],
            offset=\pgfkeysvalueof{/semesterplanner-lua/event/offset},
            scale_width=\pgfkeysvalueof{/semesterplanner-lua/event/scale width},
        }
    }
}
```

`terplannerLua@formattedEvent` Simply a layer above `\semesterplannerLua@event` which formats the content before passing it on. This formatting is thought to be a good formatting for lecture-like entries and is heavily stolen from [1] Takes a number of arguments:

1. title of the event

[1] https://github.com/nlschn/semesterplanner/

2. name of the speaker/lecturer
3. location (e.g. roomnumber)
4. day on which the event takes place (for valid values see the `day` pgf key above)
5. time (for valid values / formatting see the `time` pgf key above)
6. priority of the event (no special formatting needed, consider using one of `\phigh`, …
7. event code. This is passed to event-pgf unmodified and can overwrite any of the above keys. To add some arguments to tikz simply use `tikz/.append={draw=green}`
8. background color of the event
9. text color of the content

```
    \def\semesterplannerLua@formattedEvent##1##2##3##4##5##6##7##8##9{
        \semesterplannerLua@event[time=##5, day=##4, tikz={fill=##8}, ##7]
        {
            \unexpanded{
                \textcolor{##9}{
                    \textbf{##1}\\[.2em]
                    \raggedright{##2}\\[0.5em]\raggedright{##6}\raggedright{##3}
                }
            }
        }
    }
```

Short-hand macros for different events using the corresponding background color

`\lecture`

```
    \def\lecture##1##2##3##4##5##6##7{
        \semesterplannerLua@formattedEvent{##1}{##2}{##3}{##4}{##5}{##6}{##7}{lecture}{white}
    }
```

`\seminar`

```
    \def\seminar##1##2##3##4##5##6##7{ %##1=title, ##2=speaker, ##3=location, ##4=day, ##5=ti
code (tikz can eb set this way too but you must use append)
        \semesterplannerLua@formattedEvent{##1}{##2}{##3}{##4}{##5}{##6}{##7}{seminar}{white}
    }
```

`\tutorial`

```
    \def\tutorial##1##2##3##4##5##6##7{ %##1=title, ##2=speaker, ##3=location, ##4=day, ##5=t
code (tikz can eb set this way too but you must use append)
        \semesterplannerLua@formattedEvent{##1}{##2}{##3}{##4}{##5}{##6}{##7}{tutorial}{white
    }
```

`\meeting`

```
    \def\meeting##1##2##3##4##5##6##7{ %##1=title, ##2=speaker, ##3=location, ##4=day, ##5=ti
code (tikz can eb set this way too but you must use append)
        \semesterplannerLua@formattedEvent{##1}{##2}{##3}{##4}{##5}{##6}{##7}{meeting}{white}
    }
```

`\officehour`

```
    \def\officehour##1##2##3##4##5##6##7{ %##1=title, ##2=speaker, ##3=location, ##4=day, ##5
code (tikz can eb set this way too but you must use append)
        \semesterplannerLua@formattedEvent{##1}{##2}{##3}{##4}{##5}{##6}{##7}{officehour}{whi
    }

}{
```

At the end of the environment after all events have been collected, generate and output the tikz code needed to draw the timetable.

```
    \directlua{sp.draw(
        [[\pgfkeysvalueof{/semesterplanner-lua/length}]],
        [[\pgfkeysvalueof{/semesterplanner-lua/width}]])}
}
```

⟨/package⟩

2.2 semesterplanner-lua.lua

⟨*luaMain⟩

`init` Initialize global variables to remove previous values (e.g. events from the previous timetable)

`days` A string with the names of the weekdays for the header

`min` Time where the timetable should start. If empty this is calculated from the events.

`max` Time where the timetable should end. If empty this is calculated from the events.

```
function init(days, min, max)
    -- clean up first
    -- global variables
    EVENTS={}
    DAYS = days -- header with names of the days set from tex currently
    DAYSE = {"M","T","W","Th","F"}
    MIN = 25*60 -- bigger than any allowed value could be
    MAX = 0
    MIN_BYPASS = false -- weather min is fixed by the user
    MAX_BYPASS = false -- weather max is fixed by the user

    if(min == "") then
    else
        assert(min:match("^%d+"), "start time has to be an integer representing the HH*60+MM
        MIN = tonumber(min)
        MIN_BYPASS = true
    end

    if(max == "") then
    else
        assert(max:match("^%d+"), "end time has to be an integer representing the HH*60+MM of
        MAX = tonumber(max)
        MAX_BYPASS = true
    end
end
```

`addEvent` Adds the event to the `EVENTS` array after some validiy checks, modifys MIN/MAX if necessary

```
-- result are the global variables EVENTS, MIN and MAX
function addEvent(opts)
    if(not checkKeys(opts, {"time", "day", "content", "tikz"})) then
        error("missing argument")
    end

    opts.from,opts.to = dur2Int(opts.time)
    -- TODO convert day to corresponding number

    if(not MIN_BYPASS and opts.from < MIN) then MIN = opts.from end
    if(not MAX_BYPASS and opts.to   > MAX) then MAX = opts.to   end
    assert(opts.from < opts.to, "From has to be before to")

    table.insert(EVENTS, opts)
end
```

`draw` Draws the tikz-timetable with the global variables `EVENTS`, `MIN`, `MAX`, `DAYSE` and `DAYS`. In addition `length` and `width` are given as direct parameters.

```
-- parameters are all global variables
function draw(length, width)
    print("length", length)
    print("width", width)
    -- copy relevant variables for working on local copies
    local events = copy_array(EVENTS)
    local days = prepareDays(DAYS)
    local daysE = copy_array(DAYSE)
    local min, minH, max, maxH = prepareMinMax(MIN, MAX)

    assert(length:match("%d*%.?%d*"), "Length must be a valid length measured in cm")
    length = tonumber(length)

    textwidth = width

    tex.print([[\begin{tikzpicture}]])
    tex.print([[\tikzset{defStyle/.style={font=\tiny,anchor=north west,fill=blue!50,draw=blac
```

Draw the grid of the timetable along with clock and day labels

```
    -- print the tabular with the weekday headers
    tex.print(string.format(
        [[\foreach \week [count=\x from 0, evaluate=\x as \y using \x+0.5] in {%s}{ ]],
        table.concat(days, ",")
        )
    )
    tex.print(string.format(
        [[\node[anchor=south] at (\y/%d* %s, 0) {\week};]], #days, textwidth))
    tex.print(string.format(
        [[\draw (\x/%d * %s, 0cm) -- (\x/%d * %s, %dcm);]],
        #days,
        textwidth,
        #days,
        textwidth, -length
        )
    )
    tex.print("}")
    tex.print(string.format(
        [[\draw (%s, 0) -- (%s,%dcm);]],
        textwidth,
        textwidth,
        -length
        )
    )

    for i=minH,maxH do
        tex.print(string.format(
            [[\node[anchor=east] at (0,%fcm ) {%d:00};]],
            minuteToFrac(i*60,min,max)*-length, i
            )
        )
        tex.print(string.format(
            [[\draw (0,%fcm ) -- (%s,%fcm );]],
            minuteToFrac(i*60,min,max)*-length,
            textwidth,
            minuteToFrac(i*60,min,max)*-length
            )
        )
    end

```

Draw the nodes of the events

```
    local d
    local red = 0.3333 -- calculated in em from inner sep
    local red_y = 0.25 -- calculated in em
    for _,e in ipairs(events) do
        if e.from < max and e.to > min then -- only draw if event is in scope (part of the co
            if e.to   > max then e.to   = max end
            if e.from < min then e.from = min end
            d = search_array(daysE, e.day) - 1
            tex.print(string.format(
                [[\node[defStyle,text width=-%fem+%f%s/%d, text depth=%fcm-%fem, text height=
                2*red, -- text width
                e.scale_width, -- text width
                textwidth,
                #days, -- text width
                length*(e.to-e.from)/(max-min), -- text depth
                2*red+red_y, -- text depth
                red_y, -- text height
                e.tikz, -- free tikz code
                (d+e.offset)/#days, -- xcoord
                textwidth,
                minuteToFrac(e.from,min,max)*-length, -- ycoord
                e.content -- content
                )
            )
        end
    end
    tex.print([[\end{tikzpicture}]])
end
```

`searchArray` Searches an array for a given value and returns the index if found. On error `nil` is returned

```
function search_array(t, s)
    for k,v in ipairs(t) do
        if(v == s) then return k end
    end
    return nil
end

```

`minuteToFrac` Calculates at which fraction of the total duration of `max-min` the time `minute` is located

```
function minuteToFrac(minute, min, max)
    return (minute-min)/(max-min)
end
```

`prepareMinMax` Calculates the next hour of `MIN` (next before) and `MAX` (next after) and returns it (the hour) and the corresponding `min`/`max` (same in minutes)

```
function prepareMinMax(min, max)
    local minH = math.floor(min/60)
    local maxH = math.ceil(max/60)
    local min = minH*60
    local max = maxH*60
    return min, minH, max, maxH
end
```

`checkKeys` Checks if all `k`s are present in table `t`

```
function checkKeys(t, k)
    for _,x in ipairs(k) do
        if(t[x] == nil) then
            return false
        end
    end
    return true
end
```

dur2Int Takes a clock duration formatted as HH:MM-HH:MM, splits it, checks for validity and returns begin/end time in minutes

```
function dur2Int(clk)
    local f1,f2, t1,t2 = clk:match("^(%d%d?):(%d%d)-(%d%d?):(%d%d)$")
    if(f1 ~= nil and f2 ~= nil and t1 ~= nil and t2 ~= nil) then
        f1 = tonumber(f1) f2 = tonumber(f2)
        t1 = tonumber(t1) t2 = tonumber(t2)
        assert(f1 >= 0 and f1 < 24, "Hours have to be >= 0 && < 24")
        assert(f2 >= 0 and f2 < 60, "Mins have to be >= 0 && < 60")
        assert(t1 >= 0 and t1 < 24, "Hours have to be >= 0 && < 24")
        assert(t2 >= 0 and t2 < 60, "Mins have to be >= 0 && < 60")
        return f1*60 + f2, t1*60 + t2
    else
        error("clk string \"" .. clk .. "\" was no valid clock string")
    end
end
```

prepareDays Splits the comma-sep string days into an array

```
function prepareDays(days)
    local ret = {}
    for m in days:gmatch("[^,]+") do
        table.insert(ret, m)
    end
    return ret
end
```

copyArray Returns a copy of the table obj

```

function copy_array(obj)
    if type(obj) ~= 'table' then return obj end
    local res = {}
    for k, v in pairs(obj) do
        local c = copy_array(v)
        res[copy_array(k)] = c
    end
    return res
end
```

Prepare the module semesterplannerLua for exporting (only the functions that should be public)

```

semesterplannerLua = {
    init = init,
    addEvent = addEvent,
    draw = draw
}
return semesterplannerLua
⟨/luaMain⟩
```

3 Change History

v1.00
General: First public release 1

4 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

A
\addEvent 137

B

\BBB 57

C

\checkKeys 254
\copyArray 283

D

\draw . . . 152, 178, 187, 201
\dur2Int 262

E

environments:
 timetable 1, 9

F

\faBold 57
\foreach 171

I

\init 112

L

\lecture 1, 90

M

\meeting 1, 99
\minuteToFrac 244

O

\officehour 1, 102

P

\phigh 50
\plow 52
\pmandatory 49, 57
\pmid 51
\pnone 53
\prepareDays 276
\prepareMinMax 247

S

\searchArray 237
\semesterplanner@event 65
\semesterplannerLua@encircle . . . 10, 49, 50, 51, 52, 53, 54, 55, 56, 57
\semesterplannerLua@event 65, 80
\semesterplannerLua@formattedEvent 79, 91, 94, 97, 100, 103
\seminar 1, 93

T

\tba 59
\tbd 58
\teams 54
\tikzset 168
timetable (environment) 1, 9
\tiny 168
\tutorial 1, 96

W

\week 171, 176

X

\x 171, 178

Y

\y 171, 176
\youtube 56

Z

\zoom 55