The stdemo Package
for starray version 2.0

Alceu Frigeri*

February 2026

Abstract

As an example of how a package writer could use the *starray* package, this documents a demo package, *stdemo*, which defines a set of commands aiming to describe a set of activities and students' work associated with them. An example of how to use these commands is presented at the end of this document.

Contents

1 Introduction

The purpose of this is to create an example of how to use a *starray* in a complete setup. That for, this demo has a few parts: 1. A companion package *stdemo.sty*, 2. this document which documents not just the user level functions/commands, but also how the the companion package was created, with a small example (at the end) of how to use it.

> ***Note:*** About the version number, since this is "part" of *starray*, and to keep tracking simple, the same version number (from *starray*) will be used.

*https://github.com/alceu-frigeri/starray/tree/main/demo

2 Data Model

As an example, let's define two structures:

1. "Activities" (like a term project, course project, etc.) with associated elements:
   (a) name, acronym
   (b) coordinator
   (c) calendar events (presentation dates, exams...)
   (d) a check list.
2. Students, with associated date:
   (a) name, IDs, etc.
   (b) advisor and (perhaps) co-advisor.
   (c) reviewer(s).
   (d) flags, etc.

> ***Note:*** As in any "procedural language", it is advisable to carefully design the data model, since this will shape the functions which will set and use said data.
> ***Note:*** Pay attention to the use of the tildes, ~ , since those definitions will be made, most likely, in an *expl3* code régime, remember that spaces are ignored, therefore, if needed, use a tilde instead of a space.

2.1 Activity Set

For the activities one could set an "starray" as follow:

```
\starray_new:n {activity}
\starray_def_from_keyval:nn {activity} {
    name = Activity's~ name ,
    acronym = ACRO ,
    coord . struct =  {
        name = Coordinator's~ name,
        title = Coordinator's~ title ,
      } ,
    calendar . struct = {
        date = {-day-} ,
        week = {-week-} ,
        event = {-event-} ,
      } ,
    chkID = ,         %%% 'unique ID' for checklists
    chkmarked = ,     %%% This shall be a prop list of   marked itens
    chkunmarked = ,   %%% This shall be a prop list of unmarked itens
    chkref = ,        %%% This shall be a prop list of ref      itens
 }
```

Whereas, the "coord" sub-structure is for the activity's coordinator, whilst "calendar" shall (for instance) contains a list of calendar events, and, finally, the many "chk* " will be used for a "check list".

> ***Note:*** The "chkID" (and checklists). In many cases it's handy to have an unique identifier for a given structure. That can be obtained with `\starray_get_unique_ID:nN`, and to avoid calling this function time and time again, one can just store that ID as a field for later use.
> ***Note:*** Could the Coordinator's name and title be a direct property (avoiding the "coord" sub-structure)? of course, that's a matter of choice on how to model it.

2.2 Student Set

A student's structure might contain, besides student's name, work title, some flags, an advisor (and co-advisor, if needed), reviewer's list (with a provision for reviewer's grade, if needed).

Of course, one doesn't need to define a *starray* structure using `\starray_def_from_keyval:nn`, but, as in this, if the set of properties is known, it always makes for a cleaner definition.

Note: The fields/properties defaults can be anything, including usual LaTeX 2ε commands, like a `\rule` which is handy, for instance, when generating forms, e.g., if the fields are all set, a form can be created with the proper values, otherwise, it will be created with "rules" in place (no need to test if the properties were set).

```
\starray_new:n {student}
\starray_def_from_keyval:nn {student}
 {
  hash = , %% this shall be self hash (if any)
  first = ,
  last = ,
  name = \rule{\l__stdemo.name_rule_dim}{.1pt} ,
  ID    = \rule{\l__stdemo.ID_rule_dim}{.1pt} ,
  email = \rule{\l__stdemo.email_rule_dim}{.1pt} ,
  worktitle = \rule{\l__stdemo.worktitle_rule_dim}{.1pt} ,
  remarks = ,
  board-local = {~local/place~} ,
  board-date   = {~date~} ,
  board-time  = {~time~} ,
  grade = 0 ,
  flag-graded = \c_false_bool , %%% IF grade already calculated (or defined)
  flag-approved = \c_false_bool ,
  flag-coadvisor = \c_false_bool ,
  advisor . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo.name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo.name_rule_dim}{.1pt},
    title = \rule{\l__stdemo.title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo.email_rule_dim}{.1pt} ,
  } ,
  coadvisor . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo.name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo.name_rule_dim}{.1pt},
    title = \rule{\l__stdemo.title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo.email_rule_dim}{.1pt} ,
  } ,
  reviewer . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo.name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo.name_rule_dim}{.1pt},
    title = \rule{\l__stdemo.title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo.email_rule_dim}{.1pt} ,
    pointA = ,
    pointB = ,
    pointC = ,
    pointD = ,
    grade = 0 ,
    flag-set = \c_false_bool ,
  } ,
 }
```

3 Auxiliary Functions

Once the data layout is set (see 2) the next step is to define a set of (document level) functions, so the data can be initialized and used by the end user.

3.1 Generic Recovery Functions

`\DataField` `\DataField {⟨starray⟩} {⟨item⟩}`
`\DataGet` `\DataGet {⟨starray⟩} {⟨item⟩} {⟨store-var⟩}`

`\DataField` will recovery an item from any *starray*, for instance, ⟨starray⟩ might be ⟨activity⟩, ⟨activity.coord⟩, ⟨activity.calendar⟩, ⟨student⟩ or ⟨student.advisor⟩ etc. whilst ⟨item⟩ might be any corresponding field. The `\DataGet` will store said value in an auxiliary ⟨store-var⟩.

Note: None of those commands are expandable. In general it should be enough (for the end user) to just use `\DataField`, but it might be necessary to use a temporary variable, ⟨store-var⟩, allowing its use in an expandable context.

```
\NewDocumentCommand{\DataField }{mm}{
  \starray_get_prop:nn{#1}{#2}
}
```

```
\NewDocumentCommand{\DataGet}{mmm}{
  \starray_get_prop:nnN{#1}{#2}{#3}
}
```

3.2 Activity's Functions

One could define a single function to initialize all fields (using a key=val interface), but, in a more traditional approach one can set two functions to start the initialization process `\NewActivity` and `\ActivitySet`.

3.2.1 Creating and Setting an Activity's Data

`\NewActivity` `\NewActiviy {⟨act-ID⟩} {⟨acronym⟩} {⟨name⟩}`

`\NewActivity` will create a new activity term, ⟨act-ID⟩ will be it's identifier (hash).

> ***Note:*** Every time a *starray* is instantiated, up to two hashes are created: a numerical one (starting at one) and an "user defined one". In the `\NewActivity` function above, ⟨act-ID⟩ is that hash, so that instance can be later referenced by it. Of course, it must be an unique ID/hash.
>
> ***Note:*** One thing to be noticed about *starrays*: every structure has an associated internal index (iterator). When you create a new instance, this iterator always points to the newly created one, therefore, sparing the use of an explicit index in the subsequent commands.

```
\NewDocumentCommand{\NewActivity}{mmm} {
    \starray_new_term:nn  {activity}{#1}
    \starray_set_prop:nnn {activity}{acronym}{#2}
    \starray_set_prop:nnn {activity}{name}{#3}
    \starray_new_term:nn  {activity.coord}{}
    \starray_get_unique_id:nN {activity} \l__stdemo.tmpID_tl
    \starray_gset_prop:nnV {activity}{chkID} \l__stdemo.tmpID_tl
    \prop_new_linked:c {l__stdemo. \l__stdemo.tmpID_tl _chkmarked_prop}
    \prop_new_linked:c {l__stdemo. \l__stdemo.tmpID_tl _chkunmarked_prop}
    \prop_new_linked:c {l__stdemo. \l__stdemo.tmpID_tl _chkref_prop}
}
```

Similarly, one can define some functions to set the activity's coordinator. Of course, it's up to the package programmer to choose if one, two (or more) functions for this.

`\ActivitySetCoord` `\ActivitySetCoord [⟨act-ID⟩] {⟨name⟩} {⟨title⟩}`

The optional argument ⟨act-ID⟩ should refer to an already create activity, and, if not given, will use the current one.

```
\NewDocumentCommand{\ActivitySetCoord}{O{}mm}{
  \tl_if_blank:nTF {#1}
    {
      \starray_gset_prop:nnn {activity.coord}{title}{#2}
      \starray_gset_prop:nnn {activity.coord}{name}{#3}
    }
    {
      \starray_gset_prop:nnn {activity[#1].coord}{title}{#2}
      \starray_gset_prop:nnn {activity[#1].coord}{name}{#3}
    }
}
```

And the associated "Calendar Events", assuming there will be a fixed set of events (each semester/year), leaving the date to be set later on.

\ActivitySetNewEvent `\ActivitySetNewEvent [⟨act-ID⟩] {⟨event-ID⟩} {⟨description⟩}`
\ActivitySetEventDay `\ActivitySetEventDay [⟨act-ID⟩] {⟨event-ID⟩} {⟨date⟩} {⟨week⟩}`

The optional argument ⟨act-ID⟩ refers to an already create activity, and, if not given, the current one will used. ⟨event-ID⟩ can be any identifier. That way, the user can first define a set of events, and later on, set the associated dates.

```
\NewDocumentCommand{\ActivitySetNewEvent}{O{}mm}{
  \tl_if_blank:nTF {#1}
    {
      \starray_new_term:nn {activity.calendar}{#2}
      \starray_gset_prop:nnn {activity.calendar}{event}{#3}
    }
    {
      \starray_new_term:nn {activity[#1].calendar}{#2}
      \starray_gset_prop:nnn {activity[#1].calendar}{event}{#3}
    }
}
```

```
\NewDocumentCommand{\ActivitySetEventDay}{O{}mmm}{
  \tl_if_blank:nTF {#1}
    {
      \starray_gset_from_keyval:nn {activity.calendar[#2]}
        {
            date = {#3} ,
            week = {#4} ,
        }
    }
    {
      \starray_gset_from_keyval:nn {activity[#1].calendar[#2]}
        {
            date = {#3} ,
            week = {#4} ,
        }
    }
}
```

3.2.2 Check Lists

It's often desirable to have a "check list". What such list could entice is always up to debate, the idea behind the few next functions is to allow the end user to define which items such a list (as a matrix) might have.

\checkdef `\checkdef {⟨chkID⟩} {⟨chkPos⟩} {⟨chktext⟩}`

⟨chkID⟩ is just an ID to reference the check list item. ⟨chkPos⟩ will relate the item to a position in a matrix (tabular environment, see `\StudentCheckListTable`) and ⟨chktext⟩ is the (assumed) short text. The command `\checkdef` defines/create a new check item.

> ***Note:*** In the implementation below, three property lists are created (based on the activity unique ID). Two of them with the ⟨chktext⟩ marked and unmarked, and a third relating the ⟨chkID⟩ a given position ⟨chkPos⟩.

```
\NewDocumentCommand{\checkdef}{mmm}{
    \starray_get_prop:nnN {activity}{chkID}\l__stdemo.chkID_tl
    \prop_gput:cnn {l__stdemo. \l__stdemo.chkID_tl _chkmarked_prop} {#1}{\__stdemo.checkedbox:~\ #3}
    \prop_gput:cnn {l__stdemo. \l__stdemo.chkID_tl _chkunmarked_prop} {#1}{\__stdemo.uncheckedbox:~
      \ #3}
    \prop_gput:cnn {l__stdemo. \l__stdemo.chkID_tl _chkref_prop} {#2}{#1}
  }
```

\checklist `\checklist [⟨act-ID⟩] {⟨chkID-list⟩}`

This sets a list of ⟨chkID⟩s associated with the current student. Note that, since the checklist is based on an activity, the optional parameter allows to explicitly select one.

> ***Note:*** In the implementation below, the property list, associated with the student unique ID, will use as keys the ⟨chkPost⟩ from `\checkdef` and as associated value the ones from the activity's marked property list.

```
\NewDocumentCommand{\checklist}{O{}m}{
    \tl_if_blank:nF {#1}
      {  \starray_set_iter_from_hash:nn {activity}{#1}  }

    \starray_get_prop:nnN {student}  {chkID} \l__stdemo.chkIDa_tl
    \starray_get_prop:nnN {activity} {chkID} \l__stdemo.chkIDb_tl

    \clist_map_inline:nn {#2}
      {
        \prop_get:cnNT {l__stdemo. \l__stdemo.chkIDb_tl _chkref_prop} {##1} \l__stdemo.checkref_tl
          {
            \prop_get:ceN {l__stdemo. \l__stdemo.chkIDb_tl _chkmarked_prop} {\l__stdemo.checkref_tl}
       \l__stdemo.checkB_tl
            \prop_gput:cee {l__stdemo. \l__stdemo.chkIDa_tl _checklist_prop} {\l__stdemo.checkref_tl
      } {\l__stdemo.checkB_tl}
          }
      }
  }
```

`\StudentCheckListTable` `\StudentCheckListTable {⟨L-list⟩}{⟨C-list⟩}`

This will produces a checklist matrix. Both ⟨L-list⟩ and ⟨C-list⟩ shall be a comma list of "lines" and "columns". More specifically, each ⟨L-list⟩ element will be compose with each ⟨C-list⟩ element like "L_iC_j" which shall correspond to one of the ⟨chkPos⟩ elements defined with `\checkdef`.

> ***Note:*** Better said, this will produces the inner part of a table, sans the table begin/end. Also note, in the code example below, that each table line is finished with a `\\*`.

```
\NewDocumentCommand{\StudentCheckListTable}{mm}
  {
    \starray_get_prop:nnN {student}  {chkID} \l__stdemo.chkIDa_tl
    \starray_get_prop:nnN {activity} {chkID} \l__stdemo.chkIDb_tl
%    \tl_show:N \l__stdemo.chkIDa_tl
%    \tl_show:N \l__stdemo.chkIDb_tl
    {
      \prop_map_inline:cn {l__stdemo. \l__stdemo.chkIDa_tl _checklist_prop}
        { \prop_put:cnn {l__stdemo. \l__stdemo.chkIDb_tl _chkunmarked_prop} {##1}{##2}  }

      \tl_gclear:N \l__stdemo.table_tl
      \clist_map_inline:nn {#1}
        {
          \tl_gclear:N \l_tmpa_tl
          \clist_map_inline:nn {#2}
            {
             \tl_gset:Nn \l__stdemo.tmpx_tl {##1####1}
             \prop_get:ceNTF {l__stdemo. \l__stdemo.chkIDb_tl _chkunmarked_prop} {\l__stdemo.tmpx_tl
      } \l__stdemo.tmpy_tl
               {\tl_gput_right:Ne \l__stdemo.table_tl {\l_tmpa_tl~ \l__stdemo.tmpy_tl}}
               {\tl_gput_right:Ne \l__stdemo.table_tl {\l_tmpa_tl~ }}
            \tl_gset:Nn \l_tmpa_tl {~&~}
            }
          \tl_gput_right:Nn \l__stdemo.table_tl {\\*}
        }
      }
      \l__stdemo.table_tl
  }
```

3.2.3 Selecting an Activity

`\ActivitySelect` `\ActivitySelect {⟨act-ID⟩}`

This will just select an activity, identified by ⟨act-ID⟩ as the current one. So that, in the following commands, one can avoid the first, optional, argument.

```
\NewDocumentCommand{\ActivitySelect}{m}
  {
    \starray_set_iter_from_hash:nn {activity}{#1}
  }
```

3.2.4 Iterating over the Calendar Data

\ActivityCalendarIterate `\ActivityCalendarIterate {⟨code⟩}`

This is a helper function, based on `\starray_iterate_over:nn`, so that the end user is free to construct an "Event Calendar" with the (activity's) stored data. The suggested pattern is: 1. Select an activity with `\ActivitySelect`, then 2. execute the code for each item stored in the activity's calendar list.. The user is supposed to use (in ⟨code⟩) `\DataField` or `\DataGet` to retrieve and use the calendar's data.

```
\NewDocumentCommand{\ActivityCalendarIterate}{m}{
    \starray_iterate_over:nn{activity.calendar}{#1}
}
```

3.3 Student's Functions

3.3.1 Creating and Setting Student's Data

\student `\student [⟨student-hash⟩] {⟨last⟩} {⟨first⟩} {⟨ID⟩} {⟨email⟩}`
\studentremark `\studentremark {⟨remark⟩}`
\worktitle `\worktitle {⟨work-title⟩}`

As always, there are many ways to achieve this. The `\student` "creates" a new student entry, the (not so) optional parameter ⟨student-hash⟩ associates it with the given hash (otherwise one would have to keep track 'which index' correspond to a student). An auxiliary property list, for a checklist (see 3.2.2 and `\StudentCheckListTable`), is created using the student unique identifier, `\starray_get_unique_id`.

```
\NewDocumentCommand{\student}{O{}mmmm}{%
  \tl_if_blank:nTF {#1}
    {\starray_new_term:nn {student}{}}
    {\starray_new_term:nn {student}{#1}}
  \starray_gset_from_keyval:nn {student}
    {
      hash  = {#1} ,
      first = {#3} ,
      last = {#2} ,
      name = {#3~ #2} ,
      ID   = {#4} ,
      email   = {\tl_to_str:n{#5}}
    }

  \starray_get_unique_id:nNTF {student}\l__stdemo.tmpID_tl
    {}
    {}
  \starray_gset_prop:nnV {student}{chkID} \l__stdemo.tmpID_tl
  \prop_new:c {l__stdemo. \l__stdemo.tmpID_tl _checklist_prop}
}%
```

```
\NewDocumentCommand{\studentremark}{m}{
  \starray_gset_prop:nnn {student}{remarks}{#1}
}
```

```
\NewDocumentCommand{\worktitle}{m}{%%
  \starray_gset_prop:nnn {student}{worktitle}{#1}
}
```

\advisor `\advisor [⟨pre-nom⟩] {⟨last⟩} {⟨first⟩}`
\coadvisor `\coadvisor [⟨pre-nom⟩] {⟨last⟩} {⟨first⟩}`
\examiner `\examiner [⟨pre-nom⟩] {⟨last⟩} {⟨first⟩}`

Those are just some auxiliary commands to set the advisor's/coadvisor's/examiner's name. The first optional parameter ⟨pre-nom⟩ can be used as a title. With each call of those, a new *starry* term is created for the current student.

```
\NewDocumentCommand{\advisor}{O{}mm}{%%
  \starray_new_term:nn {student.advisor}{}
  \__stdemo.set_prof:nnnn {advisor}{#1}{#2}{#3}
}
```

```
\NewDocumentCommand{\coadvisor}{O{}mm}{%%
  \starray_new_term:nn {student.coadvisor}{}
  \starray_gset_prop:nnn {student}{flag-coadvisor}{\c_true_bool}
  \__stdemo.set_prof:nnnn {coadvisor}{#1}{#2}{#3}
}
```

```
\NewDocumentCommand{\examiner}{O{}mm}{%%
  \starray_new_term:nn {student.reviewer}{}
  \starray_gset_prop:nnn {student.reviewer}{flag-set}{\c_true_bool}
  \__stdemo.set_prof:nnnn {reviewer}{#1}{#2}{#3}
}%
```

```
\cs_new_protected:Npn \__stdemo.set_prof:nnnn #1#2#3#4
  {
    \tl_if_blank:nTF {#2}
      {
        \starray_gset_from_keyval:nn {student.#1}
          {
            last = {#3} ,
            first = {#4} ,
            name = {#4~ #3} ,
          }
      }
      {
        \starray_gset_from_keyval:nn {student.#1}
          {
            last = {#3} ,
            first = {#4} ,
            name = {#2~ #4~ #3} ,
          }
      }
  }
```

`\advisorinfo` `\coadvisorinfo` `\examinerinfo`

`\advisorinfo` {⟨institute⟩} {⟨title⟩} {⟨email⟩}
`\coadvisorinfo` {⟨institute⟩} {⟨title⟩} {⟨email⟩}
`\examinerinfo` {⟨institute⟩} {⟨title⟩} {⟨email⟩}

Some extra commands to set the advisor's/coadvisor's/examiner's other data. Note, though, it is assumed that these commands will be called after the respective `\advisor`, `\coadvisor` or `\examiner` command call.

```
\NewDocumentCommand{\advisorinfo}{mmm}{%%
  \__stdemo.set_prof_info:nnnn {advisor}{#1}{#2}{#3}
}%
```

```
\NewDocumentCommand{\coadvisorinfo}{mmm}{%%
  \__stdemo.set_prof_info:nnnn {coadvisor}{#1}{#2}{#3}
}%
```

```
\NewDocumentCommand{\examinerinfo}{mmm}{%%
  \__stdemo.set_prof_info:nnnn {reviewer}{#1}{#2}{#3}
}%
```

```
\cs_new_protected:Npn \__stdemo.set_prof_info:nnnn #1#2#3#4
  {
    \starray_gset_from_keyval:nn {student.#1}
      {
        institution = {#2} ,
        title = {#3} ,
        email = {\tl_to_str:n{#4}} ,
      }
  }
```

`\examinergrades`

`\examinergrades` [⟨case⟩] {⟨gradeA⟩} {⟨gradeB⟩} {⟨gradeC⟩} {⟨gradeD⟩}

Nothing much to be said, this allows to set 4 grades, per examiner. ⟨case⟩ sets how the average shall be calculated. Note that, this is supposed to be used immediately after the respective command `\examiner`, better said, before another `\examiner`, or after having selected a specific reviewer with `\studentreviewerselect` (see 3.3.2).

> ***Note:*** The `\starray_gset_prop:nne` is used to assure the stored grade will be a floating point, and not an expression to be evaluated later on.

```
\NewDocumentCommand{\examinergrades}{O{A}mmmm} {
  \tl_if_blank:nTF {#2}
    { \starray_gset_prop:nnn {student.reviewer}{grade}{0} }
    {
      \starray_gset_from_keyval:nn {student.reviewer}
        {
          pointA = #2 ,
          pointB = #3 ,
          pointC = #4 ,
          pointD = #5 ,
        }
      \str_case:nnF {#1}
        {
          {A}
            {
              \starray_gset_prop:nne {student.reviewer}{grade}
                { \fp_eval:n{round((#2  + #3 + #4 + #5) / 4 , 2 , 1) } }
            }
          {B}
            {
              \starray_gset_prop:nne {student.reviewer}{grade}
                { \fp_eval:n{round((#2  + #3 + #4) / 3 , 2 , 1) } }
            }
        }
        {
          \starray_gset_prop:nne {student.reviewer}{grade}
            { \fp_eval:n{round((#2  + #3 + #4 + #5) / 4 , 2 , 1) } }
        }

    }
  }
```

\studentgrade `\studentgrade` [⟨case⟩] {⟨min⟩}

Similarly, this will set/calculate the student's final grade based on the individual reviewers' grades, and the flag *flag-approved* (based on ⟨min⟩). The optional parameter ⟨case⟩ selects how the final grade will be calculated.

Note: The `\starray_gset_prop:nne` is used to assure the stored grade will be a floating point, and not an expression to be evaluated later on.

Note: To simplify the logic, it's assumed there are 3 reviewers (even though in some cases only two are needed). Therefore the command `\emptytermifnone` (see 3.3.4);

```
\NewDocumentCommand{\studentgrade}{O{A}m}
 { %assuming there are 3 of them...
   \starray_get_prop:nnN {student.reviewer[1]}{grade} \l__stdemo.A_tl
   \starray_get_prop:nnN {student.reviewer[2]}{grade} \l__stdemo.B_tl
   \starray_get_prop:nnN {student.reviewer[3]}{grade} \l__stdemo.C_tl
   \str_case:nnF {#1}
     {
       {A}
         {
           \starray_gset_prop:nne {student}{grade}
             {
               \fp_eval:n{round((\l__stdemo.A_tl + \l__stdemo.B_tl + \l__stdemo.C_tl) / 3 , 2 , 1)
     }
             }
         }
       {B}
         {
           \starray_gset_prop:nne {student}{grade}
             {
               \fp_eval:n{round((\l__stdemo.A_tl + \l__stdemo.B_tl) / 2 , 2 , 1) }
             }
         }
       {C}
         {
           \starray_gset_prop:nne {student}{grade}
             {
               \fp_eval:n{round(3 / (1/\l__stdemo.A_tl + 1/\l__stdemo.B_tl + 1/\l__stdemo.C_tl) , 2
      , 1) }
             }
         }
     }
     {
       \starray_gset_prop:nne {student}{grade}
         {
           \fp_eval:n{round((\l__stdemo.A_tl + \l__stdemo.B_tl + \l__stdemo.C_tl) / 3 , 2 , 1) }
         }
     }
   \starray_get_prop:nnN {student}{grade} \l__stdemo.tmp_fp
   \fp_compare:nNnTF {\l__stdemo.tmp_fp} < {#2}
     {
       \starray_gset_prop:nnn {student}{flag-approved}{\c_false_bool}
     }
     {
       \starray_gset_prop:nnn {student}{flag-approved}{\c_true_bool}
     }
   \starray_gset_prop:nnn {student}{flag-graded}{\c_true_bool}
 }
```

3.3.2 Selecting Student's Data

\studentselect `\studentselect` {⟨student-hash⟩}
\studentReviewerSelect `\studentReviewerSelect` {⟨rev-index⟩}

`\studentselect` allows to select a given student, given it's hash (note, the student index could also be used). Since the `\examiner` command (see 3.3.1) doesn't associate a hash with each examiner, one can only select one with it's index number: 1, 2 ...

```
\NewDocumentCommand{\studentselect}{m} {
    \starray_set_iter_from_hash:nn {student}{#1}
 }
```

```
\NewDocumentCommand{\studentReviewerSelect}{m} {
    \starray_set_iter:nn {student.reviewer}{#1}
 }
```

\studentcase	\studentCase {⟨if-approved⟩} {⟨if-not⟩}
\studentadvcase	\studentAdvCase {⟨if-one⟩} {⟨if-many⟩}
\studentcoadvcase	\studentCoadvCase {⟨if-set⟩} {⟨if-not⟩}
\studentreviewersetcase	\studentReviewerSetCase {⟨if-set⟩} {⟨if-not⟩}

Those are auxiliary conditionals. \studentAdvCase tests if the student has one or more than one advisors assigned. \studentCoadvCase tests if there is a co-advisor. \studentReviewerSetCase tests if the current reviewer (better said, student.reviewer) is set (not the default -empty- set). And, finally, \studentCase verifies the state of the flag flag-approved.

> ***Note:*** In the code below, one could have used \starray_get_prop: instead. The construct is meant as an example of how to use one of the few \starray_parsed_ commands, which can be used in an expandable context.

```
\NewDocumentCommand{\studentAdvCase}{mm} {
    \starray_term_parser:n{student.advisor}
    \int_compare:nNnTF {\starray_parsed_get_cnt:} > {1}
      {#1}
      {#2}
  }
```

```
\NewDocumentCommand{\studentCoadvCase}{+m+m} {
    \starray_term_parser:n{student}
    \bool_if:nTF {\starray_parsed_get_prop:n{flag-coadvisor}}
      {#1}
      {#2}
  }
```

```
\NewDocumentCommand{\studentReviewerSetCase}{mmm} {
    \starray_term_parser:n{student.reviewer[#1]}
    \bool_if:nTF {\starray_parsed_get_prop:n{flag-set}}
      {#2}
      {#3}
}
```

```
\NewDocumentCommand{\studentCase}{mm} {
    \starray_term_parser:n{student}
    \bool_if:nTF{\starray_parsed_get_prop:n{flag-approved}}
      {#1}
      {#2}
}
```

3.3.3 Iterating over Students

\studentiterate	\studentiterate {⟨code⟩}
\studentadvisoriterate	\studentadvisoriterate {⟨code⟩}

As their name implies, they are auxiliary commands to iterate over all students, \studentiterate, or all advisors of the current student, \studentadvisoriterate.

> ***Note:*** To retrieve the student's/advisor's data, the end user is supposed to use \DataField or \DataGet (see 3.1), like \DataField {student}{name} (do not use any index/hash) or \DataField {student.advisor}{name}.

```
\NewDocumentCommand{\studentiterate}{+m} {
    \starray_iterate_over:nn{student}{#1}
}
```

```
\NewDocumentCommand{\studentadvisoriterate}{+m} {
    \starray_iterate_over:nn{student.advisor}{#1}
}
```

3.3.4 Auxiliary Commands

\emptytermifnone	\emptytermifnone [⟨count⟩] {⟨struct⟩} [⟨code⟩]
\emptyfields	\emptyfields

\emptyfields creates an "empty" student (better said, with a hash: "empty") all fields remain at their default value (see 2.2). \emptytermifnone assures that there is(are) at least ⟨count⟩ (defaults to 1) (sub)structures ⟨struct⟩. Optionally, ⟨code⟩ will be executer after each instantiation (if needed) of a term defined by ⟨struct⟩.

```
\NewDocumentCommand{\emptytermifnone}{O{1}mO{}}
  {
    \__stdemo.emptyterm_if_none:nnn {#1}{#2}{#3}
  }

\cs_new_protected:Npn \__stdemo.emptyterm_if_none:nnn #1#2#3
  {
    \starray_get_cnt:nN {#2} \l_tmpa_int
    \int_while_do:nNnn {\l_tmpa_int} < {#1}
      {
        \starray_new_term:nn {#2}{}
        #3
        \starray_get_cnt:nN {#2} \l_tmpa_int
      }
  }
```

```
\NewDocumentCommand{\emptyfields}{} {
    \__stdemo.emptyfields:
  }


\cs_new_protected:Npn \__stdemo.emptyfields:
  {
    \starray_new_term:nn {student}{empty}
    \starray_new_term:nn {student.advisor}{}
    \starray_new_term:nn {student.coadvisor}{}
    \starray_new_term:nn {student.reviewer}{}
    \starray_new_term:nn {student.reviewer}{}
    \starray_new_term:nn {student.reviewer}{}
  }
```

3.3.5 Student's Lists

\studentaddtolist `\studentaddtolist {⟨list⟩}`
\studentlistsort `\studentlistsort [⟨field⟩] {⟨list⟩}`

A *starray* has an implicit order: it's instantiation sequence, which is ok, but not always. To be able the access/list the terms of it, an option is to have one (or more) associated lists. Note that those commands will just create a sequence associated with ⟨list⟩
`\studentaddtolist` will insert a student (better said, the current student's hash) to a ⟨list⟩, if the ⟨list⟩ isn't already defined, a new one will be created. `\studentlistsort` will sort the elements of ⟨list⟩ based on the value of the field ⟨field⟩ (defaults to "name").

> ***Note:*** Since ⟨list⟩ will store student's hash, the sort has to first retrieve the associated *starray* term and the "sorting field" which can be, in fact, anything associated with the student. In the example below, only the immediate fields can be used, like name, email, worktitle, but one can easily modify the code below to retried, for instance, the advisor's name.

```
\NewDocumentCommand{\studentaddtolist}{m}
  {
    \seq_if_exist:cF {l__stdemo.#1_list_seq}
      {
        \seq_new:c {l__stdemo.#1_list_seq}

        \bool_new:c {l__stdemo.#1_classified_bool}
      }
    \bool_set_false:c {l__stdemo.#1_classified_bool}
    \starray_term_parser:n {student}
    \seq_gput_right:ce {l__stdemo.#1_list_seq} {\starray_parsed_get_prop:n {hash}}
  }
```

```
\NewDocumentCommand{\studentlistsort}{O{name}m}
  {
    \bool_set_true:c {l__stdemo.#1_classified_bool}
    \__stdemo.seq_sort:nn {#2}{#1}
  }

\cs_new_protected:Npn \__stdemo.seq_sort:nn #1#2
  {
    \seq_gsort:cn {l__stdemo.#1_list_seq }
      {
        \starray_set_iter_from_hash:nn {student}{##1}
        \starray_get_prop:nnN {student}{#2} \l__stdemo.sortA_tl
        \starray_set_iter_from_hash:nn {student}{##2}
        \starray_get_prop:nnN {student}{#2} \l__stdemo.sortB_tl
        \str_compare:eNeTF { \l__stdemo.sortA_tl } > { \l__stdemo.sortB_tl }
          { \sort_return_swapped: }
          { \sort_return_same: }
      }
  }
```

`\listemptytermsifnone` `\listemptytermsifnone` {⟨list⟩}

This will iterate over all students in ⟨list⟩ assuring that each student has at leas 1 advisor and 3 examiners (reviewers). Note that, in the case of the examiners the `\examiner` command (see 3.3.1) sets the "flag-set" associated with said examiner to true. This, however, keeps the it's default value: false.

```
\NewDocumentCommand{\listemptytermsifnone}{m}
  {
   \seq_map_inline:cn {l__stdemo.#1_list_seq}
      {
        \starray_set_iter_from_hash:nn {student}{##1}

        \emptytermifnone{student.advisor}
        \emptytermifnone[3]{student.reviewer}
%        % those could be, instead
%        \__stdemo.emptyterm_if_none:nnn {1}{ student.advisor } {}
%        \__stdemo.emptyterm_if_none:nnn {3}{ student.reviewer } {}
      }
  }
```

`\studentlistiterate` `\studentlistiterate` {⟨list⟩} {⟨code⟩}

As the name implies, it will iterate over all students in ⟨list⟩ executing ⟨code⟩ for each of them. Note that, before starting the iteration, `\studentlistiterate` verifies if the list is already sorted (if not, `\studentlistsort` will be called, with it's default) and it will make sure all students have an advisor term (and examiners) by calling `\listemptytermsifnone`.

```
\NewDocumentCommand{\studentlistiterate}{m+m}
  {
    \bool_if:cF {l__stdemo.#1_classified_bool}
      { \studentlistsort{#1} }
    \listemptytermsifnone{#1}
    \seq_map_inline:cn {l__stdemo.#1_list_seq}
      {
        \starray_set_iter_from_hash:nn {student}{##1}
        #2
      }
  }
```

4 Example of Use

4.1 Setting Things up

```
\NewActivity{FW I}{fw0501}{Final Work I}
\ActivitySetCoord{Prof. Willian S.}{Final Work Coordinator}

\NewActivity{FW II}{fw0502}{Final Work II}
\ActivitySetCoord[FW II]{Prof. Karen S.}{Final Work Coordinator}

\NewActivity{TN A}{tn0101}{Trainees}
\ActivitySetCoord{Prof. Samantha S.}{Trainee Program Coordinator}
```

4.2 Defining per Activity Check List

```
\ActivitySelect{FW I}

\checkdef{L1C1}{docs}{Documentation OK}
\checkdef{L2C1}{prop}{Proposal OK}
\checkdef{L3C1}{advisor}{Advisor assig.}

\checkdef{L1C2}{middle}{middle term}
\checkdef{L2C2}{examiners}{Examiners assig.}

\checkdef{L1C3}{final}{Final Text}
\checkdef{L2C3}{tutorok}{Tutor approval}

\checkdef{L1C4}{text}{text approved}
\checkdef{L2C4}{graded}{examiners grade}

\checkdef{L3C5}{library}{Text Catalogued}
```

```
\ActivitySelect{FW II}

\checkdef{L1C1}{docs}{Documentation OK}
\checkdef{L2C1}{prop}{Proposal OK}
\checkdef{L3C1}{advisor}{Advisor assig.}

\checkdef{L1C2}{middle}{middle term}
\checkdef{L2C2}{examiners}{Examiners assig.}

\checkdef{L1C3}{final}{Final Text}
\checkdef{L2C3}{tutorok}{Tutor approval}

\checkdef{L1C4}{text}{text approved}
\checkdef{L2C4}{graded}{examiners grade}

\checkdef{L3C5}{library}{Text Catalogued}
```

```
\ActivitySelect{TN A}
\checkdef{L1C1}{docs}{Documentation OK}

\checkdef{L2C1}{tutor}{tutor assigned}

\checkdef{L1C2}{middle}{middle term report}

\checkdef{L1C3}{final}{Final Report}
\checkdef{L2C3}{tutorok}{tutor approval}

\checkdef{L1C4}{text}{text approved}

\checkdef{L3C5}{library}{Report Catalogued}
```

4.3 Defining per Activity Calendar

Setting a set of events related to an activity, and then the relevant dates.

```
\ActivitySelect{FW II}

%%%%%%%
\ActivitySetNewEvent{opening}
    {First class. Activity goals.}

\ActivitySetNewEvent{proposals}
    {Deadline for proposals submission.}

\ActivitySetNewEvent{middle term}
    {Students middle term presentation and follow-up.  }

\ActivitySetNewEvent{submission}
    {Deadline for student's work submission.}

\ActivitySetNewEvent{review}
    {Work review by examiners.}

\ActivitySetNewEvent{feedback}
    {Feedback on the work done, per student (private)}

\ActivitySetNewEvent{exam}
    {Final Exam}
```

Once the events are defined, the associated dates can be set, later on, at any time. There is no need, per-se, to set the dates right away.

```
\ActivitySelect{FW II}

%%%%%%%
\ActivitySetEventDay {opening}{01/09}{1st week}
\ActivitySetEventDay {proposals}{15/09}{3rd week}
\ActivitySetEventDay {middle term}{07/10}{6th week}
\ActivitySetEventDay {submission}{21/10}{8th week}
\ActivitySetEventDay {review}{28/10-03/11}{9th week}
\ActivitySetEventDay {feedback}{06-10/11}{10th week}
\ActivitySetEventDay {exam}{15/11}{last week}
```

4.4 Constructing an Activity Calendar

Once all set, it is just a matter of using the `\ActivityCalendarIterate` to go over all the events. Note that, the sequence is exactly the original one (defined by `\ActivitySetNewEvent`).

```
\ActivitySelect{FW II}
\begin{center}
  \sc Calendar for \DataField{activity}{name}
\end{center}
  \begin{tabular}{lcl}
    \ActivityCalendarIterate{ \DataField{activity.calendar}{date} & \DataField{activity.calendar}{
      week} & \DataField{activity.calendar}{event} \\ }
%    \ActivityCalendarIterate{ {date} & {week} & {event} \\ }
  \end{tabular}
```

Calendar for Final Work II

01/09	1st week	First class. Activity goals.
15/09	3rd week	Deadline for proposals submission.
07/10	6th week	Students middle term presentation and follow-up.
21/10	8th week	Deadline for student's work submission.
28/10-03/11	9th week	Work review by examiners.
06-10/11	10th week	Feedback on the work done, per student (private)
15/11	last week	Final Exam

4.5 Students

Setting student's data (some others, not shown, likely defined).

```
\student[James S.]{Smith}{James}{ID001}{smith.james@uni.gov}
\studentremark{2nd time}
\worktitle{Some Useful System}

\advisor{T.}{Jonathan}
\advisorinfo{University of Z}{Prof.}{jon.t@uni.gov}

\examiner{T.}{William}
\examinergrades{10}{9}{8}{7}
\examinerinfo{University of Z}{Prof.}{william.t@uni.gov}

\examiner{T.}{Jame}
\examinerinfo{University of Z}{Prof.}{jame@uni.gov}
\examinergrades{10}{9}{8}{7}

\examiner{T.}{Thomaz}
\examinerinfo{University of Z}{Prof.}{thomaz.t@uni.gov}
\examinergrades{10}{9}{8}{7}

\ActivitySelect{FW I}
\checklist{docs,prop,advisor,middle,examiners,text,graded}
\studentaddtolist{FW-I}
```

```
\student[Sarah S.]{Barnes}{James}{ID003}{sarah.james@uni.gov}
\worktitle{Some Useful System}

\advisor{T.}{Jonathan}
\advisorinfo{University of Z}{Prof.}{jon.t@uni.gov}

\examiner{T.}{William}
\examinergrades{0}{9}{8}{7}
\examinerinfo{University of Z}{Prof.}{william.t@uni.gov}

\examiner{T.}{Jame}
\examinerinfo{University of Z}{Prof.}{jame@uni.gov}
\examinergrades{5}{9}{8}{7}

\examiner{T.}{Thomaz}
\examinerinfo{University of Z}{Prof.}{thomaz.t@uni.gov}
\examinergrades{10}{9}{8}{7}

\ActivitySelect{FW II}
\checklist{docs,prop,advisor,middle,examiners,text,graded,library}
\studentaddtolist{FW-II}
```

4.6 Accessing a single student data

Just an example of how to select/access the data of a single student, and it's corresponding check list (associated with a given activity).

```
\ActivitySelect{FW I}
\studentselect{James S.}
Student's Name: \DataField{student}{name}

Activity: \DataField{activity}{name}

\begingroup
  \scriptsize
  \begin{tabular}{lllll}
  \StudentCheckListTable{L1,L2,L3}{C1,C2,C3,C4,C5}
  \end{tabular}
\endgroup
```

Student's Name: James Smith
Activity: Final Work I

✓ Documentation OK	✓ middle term	☐ Final Text	✓ text approved	
✓ Proposal OK	✓ Examiners assig.	☐ Tutor approval	✓ examiners grade	
✓ Advisor assig.				☐ Text Catalogued

4.7 Iterating over a list of students

Given a student list, one can just iterate over it. Note that no explicit index/hash has been used. Also note that, since the `\studentlistiterate` executes `\listemptytermsifnone`, there is no explicit need to test if there are 3 examiners.

```
\studentlistsort[last]{FW-II} % sorting the students by their last name.
\studentlistiterate{FW-II}{
    \studentgrade[A]{6} %using the A (formula) and setting the minimum at 6.
    \par
    Student: \DataField{student}{name}\par
    \studentAdvCase
      { %more than one
        Advisors:
        \studentadvisoriterate
          {
            \DataField{student.advisor}{name}, ~
          }
      }
      { %just one
        Advisor: \DataField{student.advisor}{name}
      }
    \par

    \begin{tabular}{@{\hspace{10mm}}lll}
    \studentReviewerSelect{1}
    1st examiner: \DataField{student.reviewer}{name} & grade: \DataField{student.reviewer}{grade} \\
    \studentReviewerSelect{2}
    2nd examiner: \DataField{student.reviewer}{name} & grade: \DataField{student.reviewer}{grade} \\
    \studentReviewerSelect{3}
    3rd examiner: \DataField{student.reviewer}{name} & grade: \DataField{student.reviewer}{grade} \\
    Average: \DataField{student}{grade} - \studentCase{Approved}{Failed}
    \end{tabular}

    \begingroup
      \scriptsize
      \begin{tabular}{lllll}
      \StudentCheckListTable{L1,L2,L3}{C1,C2,C3,C4,C5}
      \end{tabular}
    \endgroup
    \\[5mm]
 }
```

Student: James Barnes
Advisor: Jonathan T.
1st examiner: William T. grade: 6
2nd examiner: Jame T. grade: 7.25
3rd examiner: Thomaz T. grade: 8.5
Average: 7.25 - Approved

✓ Documentation OK	✓ middle term	☐ Final Text	✓ text approved	
✓ Proposal OK	✓ Examiners assig.	☐ Tutor approval	✓ examiners grade	
✓ Advisor assig.				✓ Text Catalogued

Student: Barney Smith
Advisor: J.J. T.
1st examiner: Ceasare T. grade: 7.5
2nd examiner: Marcus T. grade: 7.25
3rd examiner: Brutus T. grade: 8.5
Average: 7.75 - Approved

✓ Documentation OK	☐ middle term	☐ Final Text	☐ text approved	
✓ Proposal OK	☐ Examiners assig.	☐ Tutor approval	☐ examiners grade	
☐ Advisor assig.				☐ Text Catalogued

Student: Kate Smithson
Advisors: Jonathan T., William T.,
1st examiner: T. Aurelius grade: 7.25
2nd examiner: Cicero T. grade: 8.5
3rd examiner: _____ grade: 0
Average: 5.25 - Failed

✓ Documentation OK	☐ middle term	☐ Final Text	☐ text approved	
✓ Proposal OK	☐ Examiners assig.	☐ Tutor approval	☐ examiners grade	
☐ Advisor assig.				☐ Text Catalogued
