ISBN: 978-989-8533-88-3 © 2019

LIASCRIPT: A DOMAIN-SPECIFIC-LANGUAGE FOR INTERACTIVE ONLINE COURSES

André Dietrich
Otto-von-Guericke-Universität, Magdeburg / Germany

ABSTRACT

LiaScript is an attempt to enable everyone to create free and interactive online courses, without the need of being an experienced programmer. Instead, it aims to bring both parties, software- and course-developers, closer together by introducing Open-Source techniques into the Open-courSe development process. LiaScript was designed to be compatible to Common-Markdown, but it introduces lots of language extensions that deal with quizzes, surveys, ASCII-art, text2speech, animations, online programming, the integration of JavaScript, etc. as well as its own macro-system that simplifies tedious and repetitive tasks. It comes along with its own just-in-time compiler that runs in the browser and therefor does not require additional tooling.

KEYWORDS

Markdown, DSL, eLearning, Online-Course, OER

1. INTRODUCTION[1]

LiaScript was initially developed within the *"Industrial eLab-Project[2]"*, which aims to make university hardware and laboratories accessible via the Internet. But, I soon realized that only by giving remote access to these resources via a fancy website I will run into problems. The mobile Arduino-Bots could be used to teach programming, sensing, navigation, dive into operating systems or even to apply artificial intelligence. Thus, the real problem was to develop an extendable and adaptable system for creating courses (instead of a single Web-App) with different objectives and for students with different backgrounds.

Surely, creating an online-course from scratch requires a lot of expertise in different web technologies at front-end (e.g., HTML, JavaScript, CSS, testing), back-end (e.g., webservers, databases), and different communication standards to connect both sides (e.g., CRUD, websockets, AJAX). Hence, it is nearly impossible for a non-programmer to understand all of these issues, before starting to develop his or her own online course. Screen- or podcasts are not a real alternative, since they are expensive and time-consuming in production, not easy to change or translate, and require additional skills in movie cutting. That is why platforms such as Udacity or Coursea invest a lot of effort and money in high-quality course productions, which is comparable to movie productions, including screenplays, actors, different sets and locations.

Fortunately, there also exist so called Learning Management Systems that try to ease the course development (Dobre, 2015). But how is such a kind of "simplicity" achieved? Mostly, by offering integrated configuration-systems, editors and authoring-tools, that shall enable the user to create a course with a lot of buttons and menus, sub-sub-menus, and masks, whose only purpose is to hide the non-intuitive syntax and semantics of a language that can be easily interpreted machines, not by humans.

We should instead start to create languages that can be easy understood and applied by humans, in order to describe their intentions, and let the machine rack its CPU in order to find an appropriate translation.

1 See the LiaScript version at: https://liascript.github.io/course/?https://raw.githubusercontent.com/andre-dietrich/e-Learning-2019/master/README.md#1
2 Project-site: http://www.elab.ovgu.de

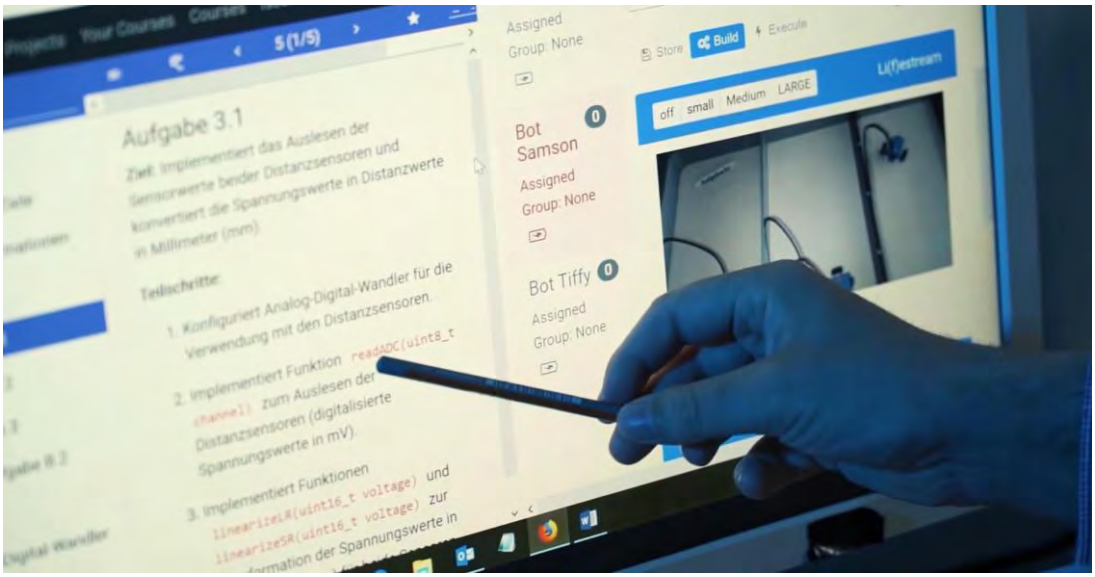


Figure 1. Picture of the eLab website with LiaScript and remote robots

2. ADAPTING OPEN-SOURCE DEVELOPMENT

More or less, all currently used systems have drawbacks in some of the following points:
Requirements from an Open-Source perspective

1. No support for larger course developer-teams, including those who develop additional functionality and those who provide content
2. No versioning, in contrast to Wikipedia as a single source of truth, content shall be provided in different "styles" for heterogeneous groups of students
3. No re-usability, parts of one course cannot simply be applied of copied into another project
4. No support for internationalization/localization (i18n), thus a course cannot simply be translated into another language
5. No variance in representation
6. Difficulties in adopting and integrating new web technologies

Pin-points 1 and 2 can be easily solved by applying a purely text-based approach for the course development and version control systems[3] (Zhou et al. 2018). All required resources, including images, videos, data-sheets, JavaScript and CSS, and everything else can be easily uploaded and made available via the internet.

2.1 Why Markdown?

Markdown (Wikipedia, 2019) is a simple meta-markup language used to structure and annotate simple text documents. Its goal is to keep the source text easy to read and write, that is why it has become more or less the standard documentation-format for Open-Source projects. Originally, it was developed to write HTML content efficiently, without having to use a WYSIWYG[4] editor. Directly writing a markup language such as HTML is considered too error prone and annoying for the writing process. Of course, I am not the first who applies Markdown to ship educational contents, earlier examples are:

3 E.g. Git (https://git-scm.com) and its web-based hosting services https://github.com or https://gitlab.com.
4 Stands for: *What You See Is What You Get*

- GitBook/Pandoc: free Markdown parser that have been widely applied in OER generation (Ovadia, 2019)
- elearn-js: converter for Markdown documents into responsive OER websites, which allows integrating quizzes, interactive images, videos, etc. (Heinecke, 2016)
- Iodide: Jupyter Notebooks brought to the internet, next to OER it can be interpreted as an example of literate programming

And of course there are other approaches that have to be mentioned (McKiernan, 2017), but the commonality of all system is that it's about creating static documents, which, although it is translated into a more beautiful format, still have to be read. To my knowledge, this approach is the only one that deals with the creation of interactive presentations, which are still generated from simple and static Markdown documents.

2.2 What is LiaScript

In contrast to other Markdown compilers that generate static HTML, LiaScript is an interpreter that downloads and renders the original Markdown document directly within the browser. That means, if the Markdown document is updated, the resulting representation will be updated too. Thus, there is no need for additional tooling, compiling steps, or server-side support. LiaScript was implemented from scratch with Elm[5] for efficiency and speed, which includes its own parser and run-time environment.

- Online interpreter, that runs directly within the browser
- Written in Elm
- Support for different representation styles (see Sec. 2.3.5)
- New Markdown language features: *Quizzes*, *Coding*, *Animations*, *Multimedia*, *ASCII-art*, …

One of the design goals was to support different rendering modes, which cover the traditional textbook mode, next to presentations with animations and spoken text. Furthermore, language itself was extended with various features, that should transform Markdown from a traditionally static markup approach into something new, suitable for interactive online courses and more.

2.3 Extensions to Markdown

Why does Markdown only support static content?

We came a long way from written scrolls to printed books to electronic books, which can still be printed out or copied by hand! But, actually it is the same old format that has been brought to a new device. Although a computer and the Internet give us much more opportunities for visualization, interaction, and story telling ...

2.3.1 Multimedia

Markdown supports 2 types of links (onto internal and external resources), which can be either direct or formatted:

```
* Direct reference: https://LiaScript.github.io
* Formatted reference: [Link to LiaScript](https://LiaScript.github.io)
```

Images can be included via formatted references that start with an exclamation mark:

```
![School in the year 2000](https://...France_in_XXI_Century._School.jpg)<!--width="100%"-->
```

In contrast to this, it is still complicated to include multimedia content. Based on the previous notation, it is possible in LiaScript to mark a link as an audio-file by adding a starting question-mark, which can be interpreted as an ear.

```
?[Joy to the world](http://www.orangefreesounds.com/...18/11/Joy-to-the-world-song.mp3?_=1)
```

5 Elm is a functional programming language that compiles to JavaScript, see the project-website: https://elm-lang.org

Due to the combination of images and sound, it is possible to insert videos. One of the benefits that lays in this notation is that every common Markdown-viewer will still generate a fully working link to these resources.

```
!?[The eLab-Project on YouTube](https://www.youtube.com/embed/bICfKRyKTwE)<!--
  width="560px" height="315px" -->
```

You might probably have noticed, that some examples contain HTML-comments with additional annotations. This is a cheap way of formatting any kind of elements, such as text, images, tables, videos, etc. A starting comment defines the format of the entire block, whereby a trailing comment changes the format of its predecessor only.

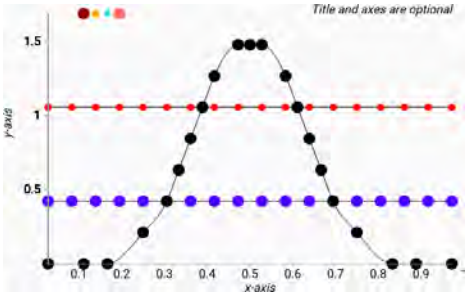
```
<!--style="color: red"-->
This whole paragraph will be colored in red ;-)<!--
class="animated infinite bounce" style="animation-delay: 3s;" -->,
but only this smiley is going to start jumping after 3 seconds.
```

This way it is even possible to define complex animation-sequences, while the content remains readable with another Markdown-viewer, since they tend to ignore comments.

2.3.2 ASCII-Art 1

From my experience, I know that a lot of produced images represent simple diagrams that represent functions, signal waves, trends, etc. These have to be generated with Excel, Gnuplot, Matlab or other tools, and to be exported, which makes it also difficult to change them or to translate labels. LiaScript offers the opportunity to draw diagrams directly within the document. Such diagrams can be easily updated and it is not necessary to switch to other "tools". The example below is turned into a nicely rendered diagram, wherein the color and the size of dots is defined by the original characters.

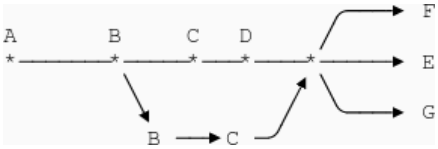
```
              Title and axes are optional
1.9 |    DotS
    |                  ***
  y |                *     *
  - | r r r r r r r*r r r r*r r r r r r r
  a |              *         *
  x |            *             *
  i | B B B B B * B B B B B B * B B B B B
  s |         *                   *
    | *  * *                          * *  *
    +--------------------------------------
    0              x-axis                1
```



2.3.3 ASCII-Art 2

If necessary, it is also possible to depict complex issues, such as graphs, UML-diagrams, or even pictures with the help of simple characters. Additionally, it is also possible to use any kind of Unicode-character. Thus, LiaScript has support for Chinese, Greek, Arabic, or any other kind of characters and symbols.

```
                                    .---->  F
A         B       C     D          /
*---------*-------*-----*--------*------>  E
           \                    ^  \
            v                  /    '---->  G
             B ------> C ---'
```

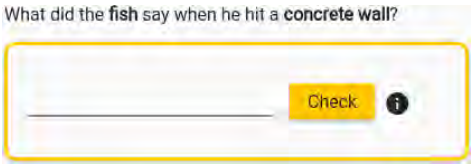


2.3.4 Quizzes

One of the language feature I wanted the most, was an easy way to integrate quizzes in different flavors and thus, to give learners the possibility to check their knowledge. Quizzes are always associated with double brackets, such that to add a text quiz, you only have to enter the solution in double braces and the input field, check-, and resolve-buttons are generated automatically.

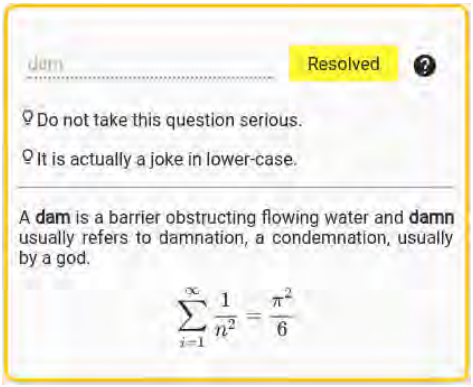
ISBN: 978-989-8533-88-3 © 2019

```
.What did the **fish** say when he hit a
**concrete wall**?

    [[dam]]
```



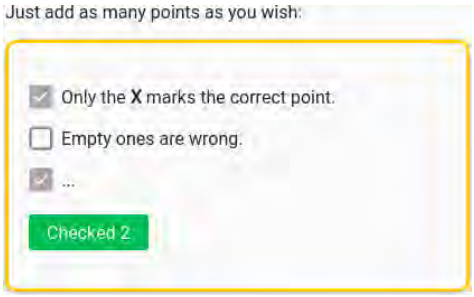
Some might adapt the question to handle the ambiguity in this case. But let us try out what LiaScript has to offer. It is either possible to add hints, by adding question-marks in double brackets and let the user decide if he needs help, by clicking onto the associated button in the rendered course. The optional `script`-tag allows to check the input, in this case to trim it and to transform it to lower-case and finally to compare it with different possible solution. Therefor the `@input`-macro gets replaced by the current user input. The trailing Markdown-blocks surrounded by two lines of stars show a more detailed explanation, which appears either if the user input was correct or if the user clicked onto the resolve button.

```
[[dam]]
_[[?]]   Do not take this question serious.
[[?]]   Do not take this question serious.
[[?]]   …
<script>
  let input = "@input".trim().toLowerCase();
  input == "damn" || input == "dam";
</script>
**********************************************
A __dam__ is a barrier obstructing flowing
water and __damn__ usually refers to
damnation, a condemnation, usually by a god.

$$
   \sum_{i=1}^\infty\frac{1}{n^2}
        =\frac{\pi^2}{6}
$$
**********************************************
```

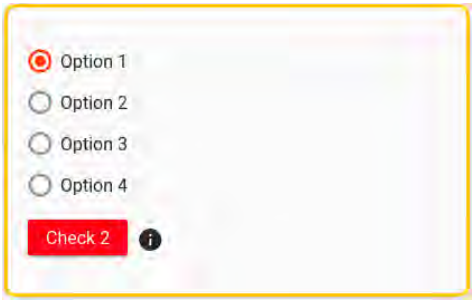


How would you encode a multiple-choice quiz with a typewriter, probably similarly to the example below. It looks like a list of simple check-buttons that define the solution. You can add as much rows/options and of course add also hints, scripts, or an explanation. A single-choice quiz is defined by stylized radio buttons, where the `X` marks the right solution and only one line is allowed to contain the `X` contain. Extensions with hints, solutions, or JavaScript checks can also be applied.

```
**Just add as many points as you wish:**
[[X]] Only the **X** marks the correct point.
[[ ]] Empty ones are wrong.
[[X]] …
```




```
[( )] Option 1
[( )] Option 2
[(X)] Option 3
[( )] Option 4
```



A generic quiz can be defined with the help of an exclamation-mark in double brackets and a script-tag. In this case a random-number is used to generate the outcome. Additional HTML-elements might be required to define different input possibilities.

```
[[!]]
<script> Math.random() < 0.2; </script>
```

2.3.5 Output Modes

The following two parts are intended to explain, how a section can be divided into several fragments and how speech output can be generated. If you are using the online rendered version of this course at LiaScript website, then you will probably have noticed that there is a button in the upper right corner. It allows switching between three different display modes. The user can decide, if he or she rather wants to listen to the explanatory text as in a presentation or to read it like a book.

1. Presentation (with voice output)
2. Slides (with notes ...)
3. Textbook (without fragmentation and voice output)

Fragmentation

Defining fragments and revealing them step by step is quite simple, only a number in double braces has to be put in front of a Markdown-block. Fragments with only one number will sustain until the end of a slide. A point of disappearance can be defined by putting a minus and a second number into double braces. Inline fragments can be defined by putting the fragment number and the elements to appear into successive braces. And it is also possible to attach different blocks to the same fragment number, either by attaching a number to every block or by putting them into a body of stars, as it was done with the solution in Sec. 2.3.4.

```
                {{1}}
This text will appear at {3}{__inline is also possible__}first.

{{2-4}} This block will appear afterwards and
disappear at step number 4.

I am here to the end of slide ...

{{4}} I come at last.
```

Voice Output

Voice output is implemented with the help of ResponsiveVoice[6]. Within an initial `comment`-tag at the beginning of the document, it is possible to define the default voice for a course. This voice can be changed at every section as well as within every voice comment itself. Such comments are treated as extended fragments, which are used explain certain fragments in more detail. Therefore, they are defined similar to block-fragments surrounded by minuses, whereby only paragraphs are allowed to follow. Depending on the presentation mode, these texts are either displayed within the slide or read out loud. This way it is also possible to implement dialogs between different persons.

```
<!--
narrator: US English Female
-->

--{{1}}-- This text is spoken by a female.

     --{{2 UK English Male}}--
I should speak with a UK like accent.

--{{3 Russian Female}}-- Я говорю по-русски с женским голосом.
```

2.4 Extending LiaScript

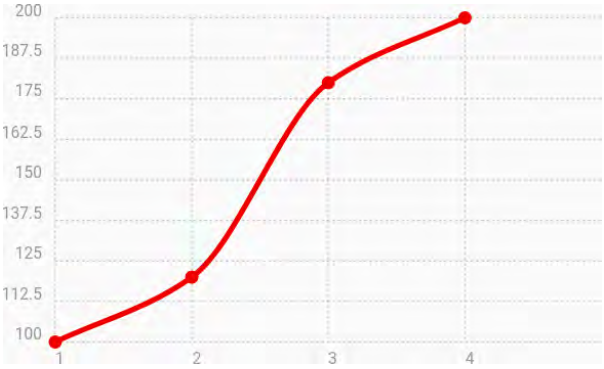
In the previous sections syntactic extensions to Markdown have been presented. But the Internet is full libraries and possibilities that a might be necessary for a particular course or topic. In contrast to many other Markdown-interpreters LiaScript allows integrating JavaScript, HTML, and CSS directly within the document. Additionally, it has support for a macro-notation that enables automating and thus simplifies repetitive and tedious tasks.

6 http://responsivevoice.org

2.4.1 ... with JavaScript and HTML

It is possible use HTML everywhere and if you want to make use of a certain JavaScript library or CSS-file, their URLs have to be included in the main comment-tag at first. Using the keyword script followed by a colon and a URL or multiple URLs, JavaScript can be integrated and similarly by using the keyword link style-sheets can be loaded. Afterwards it is possible everywhere to access the new functionality. The following example depicts, how the JavaScript library Chartist[7] is used to plot a certain graph.

```
<!--
script: https://ajax.goog...3/jquery.min.js
        https://cdn.jsdel...chartist.min.js

link:   https://cdn.jsde...chartist.min.css
-->
...
<div class="ct-chart" id="chart">
</div>
<script>
  let chart = new Chartist.Line('#chart', {
    labels: [1, 2, 3, 4],
    series: [[100, 120, 180, 200]]
  });
</script>
```



2.4.2 ... with Macros

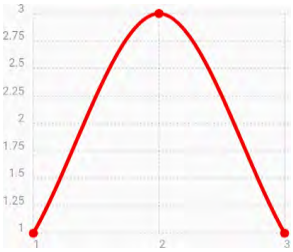
In Sec. 2.3.4, the `@input` macro has already been used to mark the replacement for the user input. A macro always starts with an `@` symbol and can be defined in the "main" comment of a document. Macros describe simple rules for text replacement. For the one-line `@red` macro, everything following the colon defines the replacement text. Parameter substitutions are defined by a `@` symbol followed by a number. These extensions can then be used arbitrarily in the document, as shown in the following example.

```
<!--@red: <b style="color: red"> @0</b>
-->
> This is a block-citation contains
> a @red(very important) example...
```

> This is a block-citation contains a **very important** example...

A macro can also call other macros, and more complex macros can be defined as a block consisting of multiple HTML, Markdown, or JavaScript elements. In this example the use of Chartist should be simplified by changing the ID for the `div` element and the content to be drawn is passed as the second parameter. This macro can also be called via a "function-like" notation. Since commas are used as separators for the parameters, back-ticks must be used here to pass the second parameter as an entire string. Admittedly, for very long entries, this can quickly become unreadable.

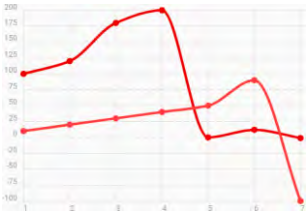
```
<!--
@Chartist
<div class="ct-chart ct-golden-section" id="chart@0">
</div>
<script>
  let chart = new Chartist.Line('#chart@0', {@1});
</script>
@end
-->
@Chartist(id1,`labels: [1,2,3], series: [[1,3,1]]`)
```



For this reason, macros can also be called within a code-block, therefor only the respective macro must be called in the head of the block. The body of the block is then passed as a single parameter. This makes it easier to define complex macros and additionally, all popular Markdown-viewers should at least display this kind of inputs in a nicely rendered code-block with syntax highlighting, which enables the interpretation of data.

7 A responsive charting library built with SVG: https://gionkunz.github.io/chartist-js/

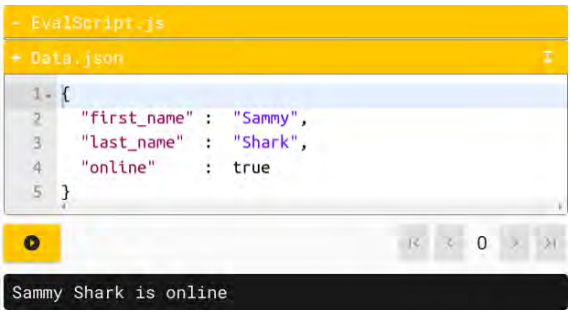
```json @Chartist(id2)
labels: [1, 2, 3, 4, 5, 6, 7],
series: [
  [100, 120, 180, 200,  0, 12,   -1],
  [10,   20,  30,  40, 50, 90, -100]]
```



2.5 Executable Code

The following syntax can be used to combine several Markdown code-blocks into one project. To the different files, titles can be associated, and they can be opened and closed. The additional `script`-tag at the end identifies these blocks as executable code and defines how the content of each block is handled. In this case the `@input` macro is called with a parameter that defines which code-block gets substituted at this position. See the LiaScript interpretation of these blocks, all files are editable and a linear version management system is used to track changes.

```` ``` js   -EvalScript.js
  let who = data.first_name+" "+
            data.last_name;
  if(data.online) who+" is online";
  else   who+" is NOT online";
```
``` json    +Data.json
  {
    "first name" :  "Sammy",
    "last_name"  :  "Shark",
    "online"     :  true
  }
```
````




```
<script>
  let data = @input(1); // insert the JSON dataset into a local variable
  eval(`@input(0)`);    // eval the script that uses the dataset
</script>
```

As shown in Sec. 2.3, it is also possible to integrate different JavaScript functionalities and libraries, so that also different programming languages can be supported. The example in Fig. 2 shows a simple *C* program that can be compiled and executed using the rextester-API[8]. The more complex definition of the associated `script`-tag was provided using the `@Rextester.eval` macro. Only by attaching such a macro, any code block can be turned into an executable one. The combination with other languages and visualizations (using HTML and JavaScript) is also possible, see the example for the programming language *Processing*. For such JavaScript libraries and also for the use of other functionalities, templates are offered that have been implemented with the help of the macro system[9]. These can be used freely and furthermore it also minimizes the breaks when reading the original Markdown document.

# 3. CONCLUSION

Looking back onto Sec. 2, points 3 to 6 have not been discussed so far. LiaScript was build around the idea of course-development as Open-Source projects. Thus, anything from one course can be used in another course, either by linking directly onto a slide or by simply copy and pasting the required parts (*req 3*). Furthermore, a growing number of templates is offered, which is founded on the internal macro-system, that ease the usage and integration of new and complex web technologies (*req 6*).

- (3.) Re-usability → anything can simply be copied into any course
- (6.) Integrating new web technologies → via a macro-system and templates
- (5.) Variance in representation → three different modes
- (4.) Internationalization/localization (i18n) → supported by specific header tags

8 An online-compile service for more than 45 different languages: https://rextester.com/main
9 List of LiaScript templates: https://github.com/liaScript/templates
```

ISBN: 978-989-8533-88-3 © 2019

LiaScript currently supports three different styles of rendering modes (see Sec. 2.3.5), allowing every user to choose his/her preferred type (req 5). Concerning the preferred course language, translating a single text-document is much easier than translating a whole software-project or a YouTube-video or podcast and LiaScript offers some options that allow to host different language versions of one course at the same project (req 4).

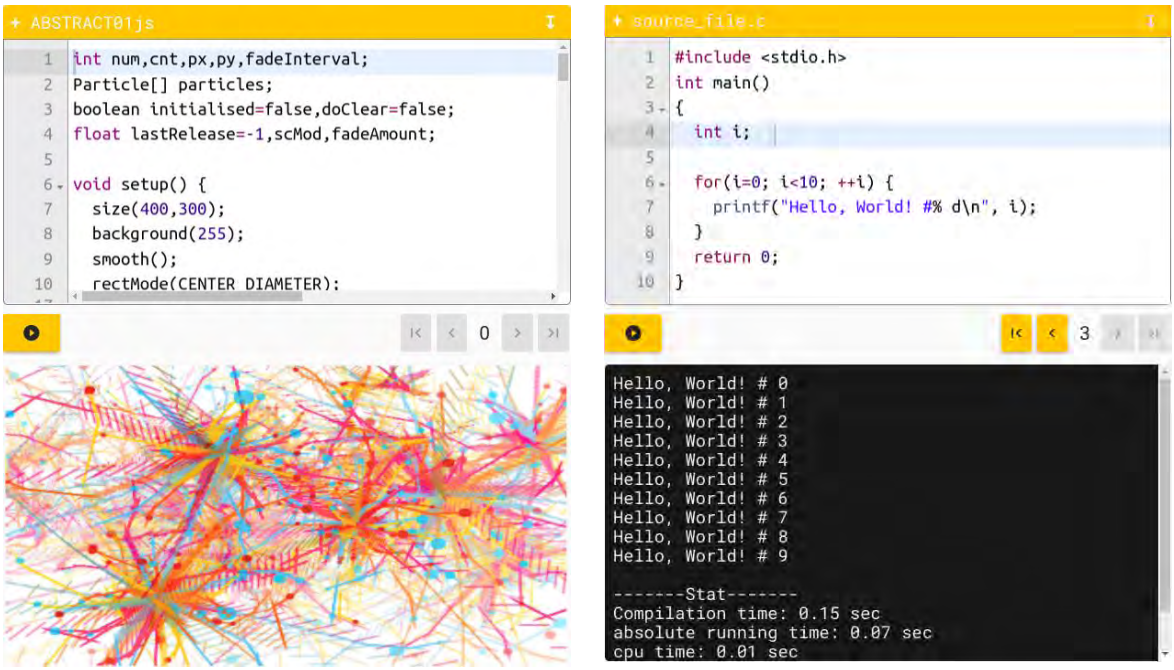



Figure 2. Programming C with rextester (left) and Processing with Prossessing.js (right)

## ACKNOWLEDGMENT

This work was partially supported by the German Federal Ministry of Education and Research (BMBF, Funding number: 16DHL1033).

## REFERENCES

Dobre, Iuliana. (2015). *Learning Management Systems for higher education-an overview of available options for Higher Education Organizations*. In: Procedia-Social and Behavioral Sciences 180, pp. 313-320.

Heinecke, Michael. (2016). *Production digitaler Skripte*. [online] Available at: http://www.sumo.uni-hamburg.de/DigitaleSkripte/ URL [Accessed Date 26 Mar. 2019].

McKiernan, Erin C. (2017). *Imagining the "open" university: Sharing scholarship to improve research and education*. In: PLoS biology, 15.10.

Ovadia, Steven. (2019). *Addressing the Technical Challenges of Open Educational Resources*. In: Portal - Libraries and the Academy, 19.1, pp. 79-93.

Wikipedia contributors. (2019). *Markdown*. [online] Available at: https://en.wikipedia.org/wiki/Markdown URL [Accessed Date 26 Mar. 2019].

Zhou, Qingguo, et al. (2018). *A collaborative and open solution for large-scale online learning*. In: Computer Applications in Engineering Education, 26.6, pp. 2266-2281.