

Project Report: SPL Programming Language

University of Puerto Rico
Mayaguez Campus
Mayaguez, Puerto Rico
Department of Electrical and Computer Engineering

SPL: A web development programming language

by

Carlos G. Rivera, Fernando Rodriguez, Ivan Miranda and Jorge Cruz

For: Dr. Wilson Rivera
Course: ICOM4036, Section 036
Date: July 1, 2017.

Project Report: SPL Programming Language

Introduction

The main objective of this project is to create a programming language that simplifies the creation of HTML documents. Through the use of commands (functions) and/or scripts ran in the terminal or command prompt, we want to allow users of this programming language to create a simple webpage intuitively, with almost no HTML programming experience required. Essentially, we want to treat elements (tags) in a webpage as objects with modifiable properties to provide customization, making web page design work in an object-oriented way. Normally, HTML documents have a lot of repetitive code, and may be messy. Our goal is to generate HTML code through these simple commands/functions, thus making simple web development less time consuming, easier, more attractive, and hopefully more effective.

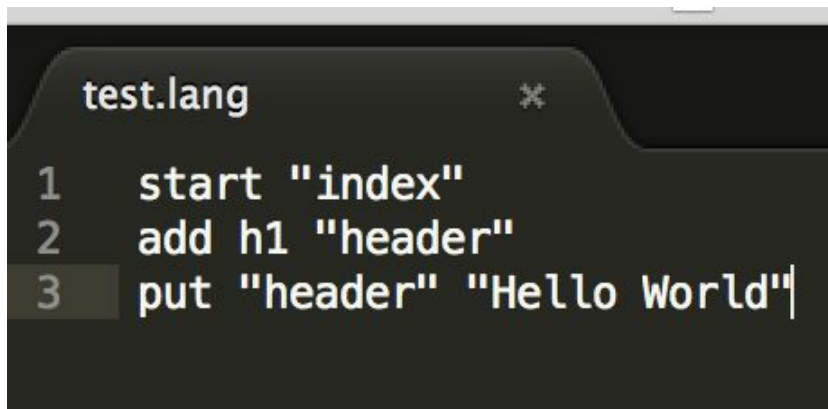
In addition, the language will provide support for more convoluted components like the ones Bootstrap provides. As such, this language will contain commands for creating blank HTML documents and various data types that will denote various HTML tags like the title tag, the header tag (with its variants), the paragraph tag, the list tag, among others and data types for Bootstrap components like tables, navigation bars, buttons, footers, SignUp boxes, Jumbotrons (emphasize information), among others. The core commands will let the user/developer create, place and fill with content (where applicable) the tags/components mentioned previously. Lastly, it will have commands for basic content styling of any tag/component, for instance to change the content's background color and font color. All content is instantiable and referenceable through an id so that all of the previous can be done without having to follow a strict order of commands and without knowing where the components are located in the document, although components (obviously) need to exist before its content and styling can be modified.

Project Report: SPL Programming Language

Language Tutorial

Getting Started

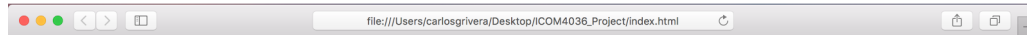
- Go to our website carlosrivera22.github.io/SPL_Website/index.html
- In our website click on the "DOWNLOAD ZIP" button to download the files needed to execute our programming language.
- Open your project on you favorite text editor like Sublime, Atom or other.
- Create a file with extension .lang, for example "test.lang"
- To start the project you must type *start "filename"* where "*filename*" is the name of the html file you want to generate.
- Let's make a simple Hello World project:
 - In test.lang type the following:

A screenshot of a code editor window with a dark background. The window title is 'test.lang'. It contains three lines of code: '1 start "index"', '2 add h1 "header"', and '3 put "header" "Hello World"'. The third line is currently selected with a light blue highlight.

```
test.lang
1  start "index"
2  add h1 "header"
3  put "header" "Hello World"
```

- In the terminal go to the project folder, like this:
 - *cd Downloads/ProjectName*
- To run the project type the following command:
 - *python3 basic.py test.lang*
- The resulting output should look like this:

Project Report: SPL Programming Language



Hello World

Congratulations! You have completed your first SPL project!

Adding HTML Components

- The following commands are used to add html components to your website project (See Language Reference Manual for more help):
 1. **add** - command that adds an html or bootstrap component to the end of the html file
 2. **addbefore** - command that adds an html or bootstrap component before another component.
 3. **addafter** - command that adds an html or bootstrap component after another component
 4. **addinside** - command that adds an html or bootstrap component inside another component
- The add command requires two parameters: name of the tag to add and its id.
- The addinside, addafter and addbefore commands require the target's id, name of the tag to add and the id of the tag to add.

Project Report: SPL Programming Language

- To add content to the html tags like paragraphs and headers you can use the *put* command.
- The put command requires the target's id and the content to add.

Adding Bootstrap Components

- The same commands we used to add html components also apply to add Bootstrap components.
- Here is an example:

```
add navbar  
{ ["Home", "home"], ["About", "about"], ["Contact", "contact"] }
```
- Go to our website to check some bootstrap components.

Styling

- To style a specific component you must use the style command with the following parameters: target component id and the style option
- Some style options consist of: width and height, padding and margins, text color and background color.
- Here is an example:

```
style "target_id" (customColorbackground:#2C3E50)
```

Images

- To add images to your project you need to use the same commands for adding any other type of component.
- The only difference is that the image component needs an extra parameter, the image's path.
- Here is an example:

```
addinside "target_id" img "image_id" "image_path"
```

Project Report: SPL Programming Language

Language Reference Manual

Commands	Parameters	Description
start	<i>filename</i>	Command that generates the html file with the specified <i>filename</i> .
add	<i>filename</i> , <i>tag</i> , <i>tag_id</i>	Command that adds an html component or <i>tag</i> to the end of the specified html file in <i>filename</i> . The new tag is referenceable as <i>tag_id</i> .
addinside	<i>target_id</i> , <i>tag</i> , <i>tag_id</i>	Command that adds an html component or <i>tag</i> inside <i>target_id</i> 's body (new nest level). The new tag is referenceable as <i>tag_id</i> .
addafter	<i>target_id</i> , <i>tag</i> , <i>tag_id</i>	Command that adds an html component or <i>tag</i> after <i>target_id</i> 's closing tag. The new tag is referenceable as <i>tag_id</i> .
addbefore	<i>target_id</i> , <i>tag</i> , <i>tag_id</i>	Command that adds an html component or <i>tag</i> before <i>target_id</i> 's opening tag. The new tag is referenceable as <i>tag_id</i> .
put	<i>target_id</i> , <i>content</i>	Command that writes the specified <i>content</i> to the specified tag by <i>target_id</i> .

Project Report: SPL Programming Language

style	<i>target_id</i> , <i>style_option</i>	Command that applies the specified <i>style_option</i> (as known for HTML programming) to the specified <i>target_id</i> 's content.
-------	---	--

HTML Component	Description
p	Tag to denote a paragraph in an HTML document.
div	Tag to denote a section in an HTML document.
h1 to h6	Tag to denote an HTML heading. h1 is the largest heading, while h6 is the smallest heading.
img	Tag to denote an image in an HTML document.

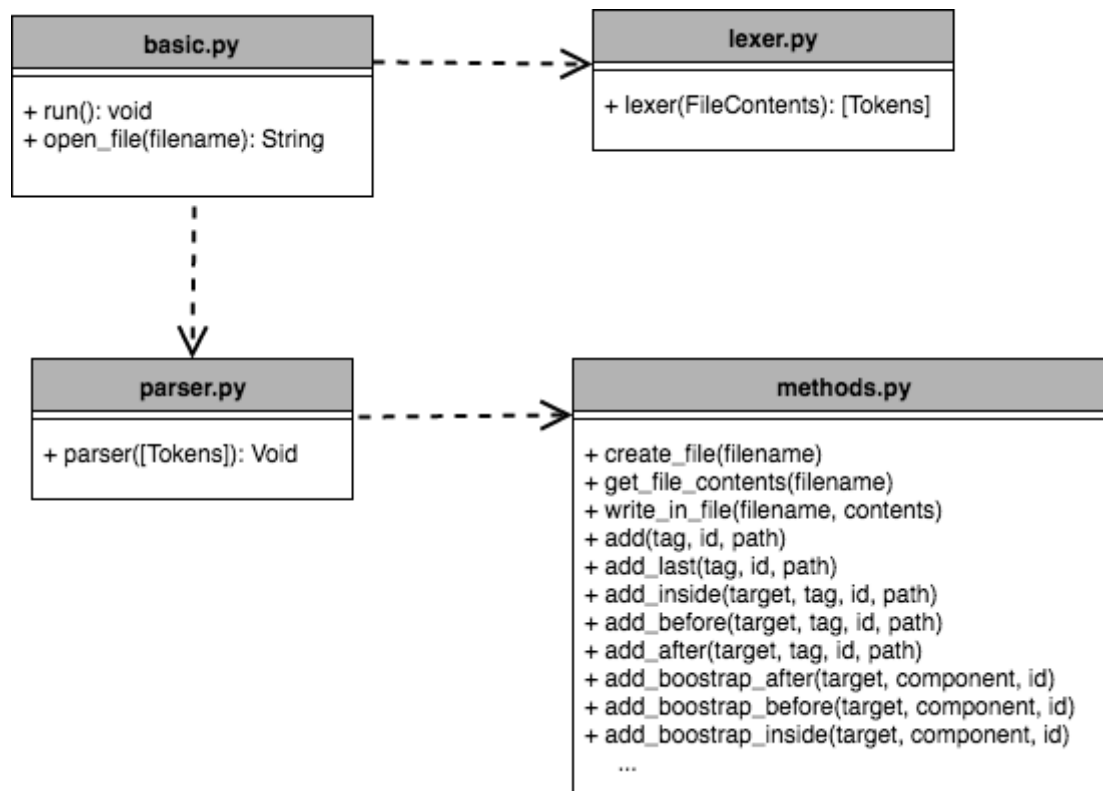
Bootstrap Component	Description
row	Bootstrap tag to denote a row.
col-size-1	Bootstrap tag to denote a column. <i>size</i> can be <i>sm</i> (for tablet view), <i>md</i> (for desktop view), or <i>lg</i> (for a larger desktop view).
table	Bootstrap tag to denote a table.

Project Report: SPL Programming Language

signup	Bootstrap tag to denote a signup form/box.
jumbotron	Bootstrap tag to denote a jumbotron: a big box that attracts extra attention to special content or information.

Language Development

Translator Architecture



Interfaces Between Modules

When the program is executed from the main module, **basic.py**, a file is opened containing the commands/code on the SPL language. This file is scanned by

Project Report: SPL Programming Language

the lexer module, `lexer.py`, effectively tokenizing the input stream into separate valid tokens that the parser may later use and analyze. In turn the parser module takes the tokens and makes sure it corresponds to the grammar of some command in the system and later executes that command. If the parser does not match a series of tokens to a command then the execution will stop all together. If there is a match it will call the corresponding function to execute the command in the functions module, `method.py`. The execution will continue until all tokens have been processed by the parser or a token or series of tokens is not matched with a command.

Software Development Environment Used On Translator

We chose to develop this using atom because it was easier to have multiple files of multiple programming languages. We had `.lang`, `.py`, `.jpg`, `.pyc` and `.html` files in our project. Other IDEs might have not supported some of these file types. We also used Sublime text which has similar features to atom.

Atom - <https://atom.io>

Sublime Text - <https://www.sublimetext.com>

Test Methodology Used During Development

To test our project's efficiency we relied on the functional testing test methodology. We basically verified that each function of the project operated in conformance with the requirements. For the testing process we had to first identify the test input or test data, which in our case where the commands and parameters of our programming language. After that we tested each component of the project separately (lexer and parser). For the lexer we verified that each valid command was properly added to the tokens array and for the parser we verified that each command was correctly processed and executed. When each unit of the project was properly tested we proceeded to test the system as a whole. This is done by interfacing the lexer and the parser and testing them with different combinations of data or input and

Project Report: SPL Programming Language

monitoring the system's behavior. The final step of our testing methodology was to make sure that the system met all the project requirements. In other words make sure that, even if the project works perfectly, the system serves its purpose.

Programs Used Testing Translator

The way we tested our translator was that we would write the code in a .lang file and run the basic.py file with the .lang file as a parameter from the terminal. The resulting file would be saved as a .html file which would then automatically launch the default browser (Safari or Google Chrome) and we would verify if we had achieved the desired webpage.

Conclusions

While it is true the main goal of the project is to develop a more intuitive way of creating webpages through html documents, it also gave us a deeper understanding of the inner workings of a programming language and how it handles all the inputs and grammar the developer writes.

Although a compiler for a robust programming language is much more complex and involves many more working parts, the lexer and parser perform a very important job, basically the pre-processing of the code. From the brainstorming sessions on how the grammar might look like, making it as natural to the end user as possible, to the manual tokenizing of commands and parameters, the whole process is very insightful into what's really happening backstage.

The end product is a simple way to create html documents with little background on programming. The intention is that a look into the documentation can easily guide you into creating a webpage, maybe a portfolio or landing page, in as little time as possible, while adding a bit more complexity with the bootstrap commands that create responsive and better looking components. By no means is it a complete language since it lacks robustness and needs a lot more commands for extra functionality but it's definitely a step in the right direction.