text++


```
Joi Anderson - jna2123      // Manager + Tester
Klarizsa Padilla - ksp2127  // Language Guru + Tester
Maria Javier -mj2729        // System Architect + Tester
```

// Introduction

text++ is a markup language designed for the production of technical documentation in an intuitive programming form. Unlike other templating languages like LaTeX, text++ is a markup language with algorithmic computing capabilities, allowing programmers to write documents as efficiently as they would write code.

text++ aims to eliminate the tedious work of repetition and copy-and-paste, by enabling the user with features like:

- Custom classes and templates for multipurpose document creation
- Control structures for simple regeneration of both text and style formats
- Reusable building blocks like primitives, variables, and objects
- Interpolation to streamline modifications

Other template processors begin with text and only apply transformations that turn the document into a formatted output. text++ not only formats text, but also performs computations in order to regenerate text and evaluates arithmetic expressions to include alongside text and document content. Integrating calculations from iterative work and control structures into text documents are simple with text++, as opposed to other static-site-generators.

Document creation tools contain features meant to help create legible writing documents and basic visuals but often involve downloading plugins that increase compile latency. The capabilities of text++ allow users to create shortcuts and quickly generate elegant documents, without relying on built-in support.

// Types

Data Types

Operator	Functionality
int	32-bit signed integer
bool	8-bit boolean variable
string	array of ASCII characters

// Operators

Arithmetic Operators

Operator	Functionality
+	addition
-	subtraction
*	multiplication
/	division
%	modulo
!	not
||	or
&&	and

Relation Operators

Operator	Functionality
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

// Control Flow

	Functionality	Syntax

If / else / elif	Conditional expression	`if (condition):`
		`  exp`
		`elif(condition):`
		`  exp`
		`else:`
		`  exp`

while / for	Iteration	`while(condition):`
		`  exp`
		
		`for(count*; condition*; update*)`
		`  exp`

continue / break	Branching control	`break; continue;`

// Built-In Functions

	Functionality	Syntax
render	Displays	`x= "text"` `render(x)`
\n	New lines	`This is a sample.\n`

table	Branching control	`table(rows, columns)`
image	Renders an image	`image(example.jpg)`
hyperlink	Auto Hyperlinks	`hyperlink(www.columbia.edu)`
indent	For 3 spaces	`indent(3) // three indents`
space	Line spacing	`line(2)  // double spacing`
date	Date stamp	`date() // returns date`
time	Time stamp	`time()`
break	Page break	`pagebreak()`
highlight	Syntax highlighting	`highlight {` `}`

// Indentation

To indicate a block of code, indent line of the block by the same amount

// Semicolon free

text++ does not use semicolons at end statements.

// Comments

// used in front of the code or text for line comment

/* */ used at the beginning and end, respectively, of code or text for block comment

// Functions

Functions can return multiple values and be defined as shown below:

```
def function_name(parameter):
     // function body
```

To call function:

```
@function_name()
```

// Templates

Templates are reusable, preformatted layouts created by the user. Templates contain static data and structures, as well as placeholders for dynamic data. Each template can be extended on any part of the document and be overridden where placeholders are defined.

.

Sytax

To define a template:

```
def template templateName   // to define a new
```

Placeholder sytax :

```
{{block placeholder_name}}
{{end placeholder_name}}
```

To extend a template:

```
{% extends templateName %}
// Write code here
{%end extends%}
```

To substitute dynamic data into placeholder fields:

```
{{% block placeholder_name %}}
{{% end placeholder_name %}}
```

// Example Code

User Defined Classes

```
def allCaps(str: String)
      return(str.toUpper)

# Welcome to @allCaps(coms 4115). // returns Welcome to COMS 4115
```

User Defined Templates

```
def template mutlipleChoice()
      # Exercise {{problem_number}}
      ##{{block prompt}}
        {{end prompt}}

      ### a. {{block optionA}}
      ### b. {{block optionB}}
      ### c. {{block optionC}}
```

Loops

```
table
      @for(int x <- 0 until 3)
            @row Homework {x}
```

Output:

Homework 1
Homework 2
Homework 3

// References

LaTeX Quick Reference
<http://www.icl.utk.edu/~mgates3/docs/latex.pdf**>**
Markdown Cheatsheet
<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet**>**
Python Flasks Templates Reference
http://flask.pocoo.org/docs/1.0/tutorial/templates/#
Python Language Reference Guide
<https://docs.python.org/2/reference/index.html
Scalatex Documentation
http://www.lihaoyi.com/Scalatex/
Twirl Reference Guide
<https://github.com/spray/twirl/blob/master/notes/about.markdown