3Lm

HAVE FUN WHILE LEARNING!

# About

3LM is a simple, interpreted language used alongside Python. It is used for defining objects and structures, intended mainly to help write structured content fast and with ease. Whether you are a scientist, a programmer or an educator, 3LM can be useful for you.

3LM is intended to be a structure language for a platform for education.

This project is under the KM brand.

# Tutorial

Following is an example 3LM file that provides a **Course** object containing two **Page** objects and an **id** attribute. The two Page objects both contain a **title** and an implicit **body** attribute.

1. Course my\_course
2. id = “e1f4b2a...”
3. ---
4. Description of course.
5. ---
6. Page
7. title = “Welcome to the first page of my course”
8. ---
9. This is the body content of the page. You can write as much as you’d like!
10. ---
11. End
12. Page
13. title = “Second page”
14. ---
15. ”Marie François Xavier Bichat” is the founder of histology.
16. ---
17. End
18. ...
19. End

Notice that the **Page** objects have no names, thus their **name** attribute is an empty string.

# LANGUAGE

## Structure

The following is the basic structure of the 3lm language:

1. Class my\_name
2. attribute = <expression>
3. body = <expression>
4. SubClass my\_sub\_name
5. ...
6. End
7. End

This is a more extended syntax:

1. <imports>
2. Class my\_name
3. attribute = <expression>
4. ---
5. Markdown text.
6. ---
7. SubClass my\_sub\_name
8. ...
9. End
10. End

**---** are opening of the body and optionally closing the body.

. . . are not typed, they only illustrate extra lines.

Notice that the language has four syntaxes. One implicit, another for expressions, and last for the body (as content).

## Syntax

A file with 3lm extension is considered as a 3lm file. By being an input file, it can import other 3lm files.

A 3lm file contains import statements at the top. The imports don’t necessarily have to state the extension since the parser automatically checks against 3lm extension:

1. import example
2. ...
3. import <name-of-file>

Next, a language file defines an unlimited number of *blocks*. A block is basically a node with optional sub-nodes.

* Blocks are defined by specifying its class or type, and optionally its name.
* Blocks can be nested inside blocks.
* In the previous examples, There’s a block with type “**Class**” and name “**my\_name**”.
* Block types are CamelCase. Starting with a capital letter and with only English letters and numbers.
* Block names are preferably snake\_case. With only English letters and numbers.

A block can have unlimited number of *attributes* that evaluates an *expression* on interpretation.

* An attribute, in the previous examples, is “**attribute**” or “**body**”.
* Attributes are preferably snake\_case. Only English letters and numbers, beginning with a letter or an underscore.
* A special attribute named **body**, can be represented in another way (see below).

A block can also have a body component if a body attribute isn’t specified.

* The body can be an attribute that evaluates an *expression*.
* It can also be started by three hyphens. Text after it is called *body content*. It can end with three other hyphens in the same indentation level of the starting ones (in a single, separate line of text).
* In the previous example, the body content is basically “**Markdown text.**”.
* The body content can be started in any indentation, the first character encountered other than whitespace is the initial content’s indentation level. Next lines should be either at this indentation level or more.
* Body attribute is tokenized and trimmed, from left-to-right, of extra indentations in the same amount of body content’s indentation.

For example, take the following:

1. Class my\_name
2. attribute = <expression>
3. ---
4. I started after 2 ignored spaces.
5. Me too.
6. I start with a space.
7. I start early.
8. My indents are none.
9. ---
10. another\_attribute = <expression>
11. SubClass my\_sub\_name
12. ...
13. End
14. End

The body would yield “I started after 2 ignored spaces.\nMe too.\n I start with a space.\nI start early.\nMy indents are none.\n”.

Notice how each line is treated by its own indentations, with regards to the initial content’s indentation level. The body content language is template markdown (using Jinja2). It supports both double curly braces and expressions within, and HTML.

The body as body content makes it easier to write and the optional three hyphens are useful for adding more attributes and blocks (such as references).

**Note that the body does not accept HTML JavaScript tags.**

## Expressions

An expression can be a string, number, function, or reference.

* **Strings**: Same as in Python. ‘string’, “””string”””, b“ByteString”.
* **Numbers**: Hex, Dec, or Oct numbers are allowed.
* **Function**: function(expression)
* **Reference**: Class:name.some\_attr, Class:this.some\_attr, this.some\_attr

A reference can refer to a function. For example, when an attribute is set to a function.

Expressions support arithmetic ops: (“” + “”), (5 / 2), (5 // 2)

Expressions support parentheses. ( expression \* ( expression + expression ))

## Markdown

Markdown being used is HTML compliant.

# Language Interpreter

The package includes a lexer and a parser. It also comes with “contrib” module that provides visualizers and interpreters. The module also includes a basic extension class with examples.

# Django Project

## Features

1. An outstanding collaboration platform with many features whether free or paid.
2. Great studying experience.
3. Command-line tool built with Python.
4. Rate limited APIs.
5. Availability of a points system (Elmas).
6. A variety of subscriptions payable by cash, credit card or by elmas (points).
7. Cross-platform apps for both Android and iOS.
8. Customizable web app options such as themes, filtering …etc.
9. Exam ques automatic extraction?
10. Collections of all your favorite stuff into categories, viewable by others based on your choice.
11. Version controlled Πs (Pies/Slides) and their sentences.
12. Courses (pie collections) are available with taxonomies and related models.
13. Searchable, modifiable, mergeable and citable sentences using VELM technologies.
14. Disclosures, notes, and references are available to add to Pies/Slides.
15. Likeliness lasting of audits for a long time in databases/files.
16. Reminders, notifications and calendar integration.
17. Messaging?

## Project Management

* Developers for iOS, Android, and C# needed alongside the frontend developer and backend developer.
* Data analysts are needed in order to do research and progress more effectively.
* Scientists and researchers are needed in order to write their content.

## E3LM TECHNOLOGIES

The “e” in e3lm is added because in most systems, you cannot start with a number.

e3lm works with Git mechanisms to enable views of edits and contributions for admins, moderators and end-users. Pie edits are sequential. When a Pie is accessed, it can be contributed to only if you’re targeting the latest edit of the Pie. The latest edit must be verified (see Verification methods for Pies) before being merged.

Each edit has the same list of related models (that are also merged) of the original Pie, which enables removing and adding new models. A Pie edit has its own discussion section, watching and voting. A pie edit cannot be collected, but rather the original reference to this Pie, including its viewable edits, are collected.