# Introduction

Dans le cadre du cours « Compilateurs », les étudiants devaient se composer en équipe afin de réaliser un projet impliquant la réalisation d’un compilateur utilisant la librairie Python PLY.

Il existe actuellement certains nombres de *frameworks* pour réaliser des présentations informatiques, notamment Reveal. L’écriture d’une telle présentation passe par un fichier HTML fastidieux à remplir à la main. Notre équipe, composée de Dany Jupille, Jason Racine et Johan Chavaillaz, a décidé de profiter l’occasion pour créer un langage simplifiant la génération d’une présentation HTML utilisant le *framework* Reveal.

Pour d’autres informations utiles concernant le projet, voir le wiki de la forge à l’adresse suivante : <https://github.com/Chavjoh/SlideMaker/wiki>

# But fixé

Comme indiqué plus haut, le but fixé par l’équipe était de réaliser un compilateur générant du code HTML pour une présentation utilisant le *framework* Reveal. Afin de ne pas complexifier la création d’un tel compilateur dès le début du projet, l’équipe a implémenté les différentes fonctionnalités de manière incrémentale.

Par cette méthode de développement, le compilateur à posséder plusieurs « états » de fonctionnement avant sa version finale :

* état 1 : génération du *header* et du *footer* de la page HTML à partir de templates ;
* état 2 : rajout de la gestion des sections du *framework* Reveal pour la génération de *slides* ;
* état 3 : rajout d’éléments basiques composants une présentation : images, listes, textes ;
* état 4 (final) : rajout de la possibilité d’importer les données d’un fichier .csv.

D’autres fonctionnalités composant ces différents états sont listées dans leur chapitre respectif.

# Langages choisis

Plusieurs langages sont impliqués dans la génération d’un code pour le *framework* Reveal :

* HTML : contient la présentation à proprement parler ;
* CSS : styles prédéfinis par le *framework* Reveal ;
* Javascript : scripts prédéfinis par le *framework* Reveal.

Le code CSS et Javascript représente le *framework* Reveal, en plus d’autres fichiers auxiliaires. Ces derniers ne changent jamais, quelle que soit la nature de la présentation. Il serait donc « stupide » de générer ce code à partir de notre compilateur, d’autant plus que ceci augmenterait considérablement sa taille.

Notre équipe a alors décidé de se contenter uniquement de la génération du fichier HTML. Le *framework* est déjà présent dans le dossier du compilateur, et celui-ci envoie les fichiers générés directement dans le bon dossier.

Au final, le compilateur traite les 2 langages suivants :

* Python : langage du compilateur, utilisation de la librairie PLY ;
* HTML : langage cible du compilateur.

# Fonctionnalités implémentées

Ce chapitre liste les différentes fonctionnalités présentes dans notre compilateur, dans leur ordre d’implémentation.

* Génération du *header* et du *footer* de la page HTML (si on compile un fichier vide)
* Génération de *slides* de présentation (à défilement vertical et/ou horizontal)
* Affichage de titres principaux (*header* des *slides*)
* Changement de la couleur de fond des *slides*
* Changement de la couleur de transition des *slides*
* Structuration des *slides* à l’aide d’un système de *layout* simple basé sur des boîtes
* Affichage d’un titre (texte plus grand que le texte normal)
* Affichage d’une image
* Affichage d’une liste à puces ou numérotée
* Affichage d’un texte simple avec opérateur de concaténation effectif (‘+’)
* Affichage d’un texte riche (texte auquel on peut appliquer du code CSS)
* Itération à l’aide d’un système de boucle similaire aux « *for* » conventionnelles
* Importation complète ou partielle d’un fichier .csv pour en afficher le contenu
* Rajout de commentaires dans le code (balises « // […] » et « /\* […] \*/ »)

# Prise en main

Ce document démontre une utilisation basique des différentes fonctionnalités du programme. Pour une meilleure prise en main, n’oubliez pas de lire les exemples !

Pour générer une présentation HTML à l’aide de notre compilateur, il suffit d’exécuter la commande suivante :

Compiler.py CHEMIN\_DU\_FICHIER

Ceci générera le fichier HTML dans le dossier « *compile* » présent avec le compilateur. Celui doit rester où il se trouve si vous souhaitez son bon fonctionnement. Quant aux chemins des images, ils doivent être indiqués par rapport au dossier « *compile* », et non par rapport au dossier où se trouve votre fichier de code !

Création d’une *slide* vide :

Slide() // Plutôt simple non ?

{ }

Création d’une *slide* verte avec le titre « Bonjour ! », avec un exemple de concaténation de texte, et une image en-dessous du titre.

Slide("background: #00ff00")

{

Title("Bonj" + "our ! "); // Concaténation inutile, mais jolie à voir

Image("bonjour.png");

}

Création d’une hiérarchie de *slides* vertes avec comme *header* « Chapitre 1 » et une transition entre les *slides* de couleur rouge :

Group("Chapitre 1", "background: #ff0000")

{

Slide("background: #00ff00")

{ }

Slide("background : #00ff00")

{ }

}

Par défaut, les éléments sont organisés de haut en bas et sont centrés. Pour les organiser de gauche à droite en plaquant le contenu en haut à gauche, on se sert de boîtes horizontales :

Slide("background: #00ff00")

{

HorizontalBox(20) // <- 20px d’espace entre les éléments

{

Text("Coucou, je suis un texte simple plaqué en haut à gauche !");

Text("Et je suis juste à sa droite !");

}

}

On dispose également de boîtes verticales (pour réorganiser de haut en bas avec le contenu plaqué en haut à gauche). Ces deux types de boîtes peuvent s’imbriquer à volonté, permettant ainsi la création d’une structure complexe.

Important : les boîtes ne doivent pas être vides ! Cela résulterait en une erreur.

Slide("background: #336633")

{

// TODO mettre un titre ici !

/\* Heureusement qu’on a la possibilité de commenter, on pourrait se perdre

dans un code pareil... ;) \*/

VerticalBox(0)

{

HorizontalBox(10)

{

// TODO mettre du texte ici !

Text("");

}

HorizontalBox(0)

{

VerticalBox(0)

{

// TODO titres aux éléments ci-dessous

Text("");

}

HorizontalBox(0)

{

// TODO éléments à insérer ici !

Text("");

}

}

}

}

Rajouter du texte riche dans une *slide* pour le colorier par exemple :

Slide()

{

/\* Les différents éléments présents dans le bloc "RichText" sont fusionnés

dans une seule et unique chaîne de caractères. \*/

RichText()

{

Element("Vert !", "color: #00ff00");

Element(" Rouge !", "color: #ff0000");

}

}

Rajouter une liste à puces ou numérotée :

Slide()

{

List("unordered") // Liste à puces

{

// Insertion de blocs "Text" et "RichText" uniquement !

}

List("ordered") // Liste numérotée

{

// Idem que ci-dessus

}

}

Utilisation d’une boucle « *for* » :

/\* Il existe deux types de blocs "For" :

1. Génération de slides (bloc "For" englobant des blocs "Slide")

2. Génération de contenu (bloc "For" englobés dans des blocs "Slide") \*/

Group("Chapitre 3.2")

{

// Génération de blocs "Slide" (ici, 5 slides dans la portée [1;5])

For(i, 1, 5)

{

Slide()

{

// On peut se servir de l’itérateur !

Title("Titre " + i);

// Génération de contenu

For(j, 1, 3)

{

Image("image" + i + j + ".bmp");

}

}

}

}

# Importation d’une table à partir des données d’un fichier .csv :

Slide()

{

/\* Un paramètre est nécessaire, les autres sont optionnels :

1. Chemin et nom du fichier .csv

...2. Chaîne contenant les 4 options disponibles \*/

Table("data.csv", "col-start:3, col-count:4, row-start:2, row-count:6");

/\* Valeurs par défaut des options :

col-start = 0

col-count = N, où N est égal au nombre de colonnes du fichier .csv

row-start = 0

row-count = N, où N est égal au nombre de lignes du fichier .csv \*/

}

# Difficultés rencontrées

Un problème plus difficile que les autres a été rencontré lors de la réalisation du compilateur. Il s’agit du rajout du bloc « *For* ». En effet, ce qu’il peut contenir est dépendant de sa position dans le programme !

Group("Chapitre 3.2")

{

// Ne peut que contenir des bloc "Slide"

For(i, 1, 5)

{

Slide()

{

// Peut contenir tous les blocs possibles dans un bloc "Slide"

For(j, 1, 3)

{

// ...

}

}

}

}

La résolution du problème s’est faite au niveau syntaxique. Même si l’utilisateur à l’impression de manipuler ces deux boucles « *for* » de manière similaires, d’un point de vue programme il s’agit de deux blocs bien distincts !

Ainsi, tant que l’on ne se trouve pas dans une slide, notre analyseur syntaxique va générer des nœuds « *sliderFor* » à chaque bloc « *For* » rencontré. En revanche, s’il en croise un à l’intérieur d’une *slide*, il va cette fois-ci générer un nœud « *contentFor* ».

Grâce à cette distinction au niveau syntaxique, notre système de boucle « *for* » est facile à tester pour vérifier que l’utilisateur n’a pas mis de mauvais blocs à l’intérieur de celles-ci.