Un mini-moteur html

Saé développement avancé

Vincent Nguyen et Florent Becker

Semestre 3, But2 22-23

1 Présentation de la Saé

Cette SAÉ vous invite à découvrir les fondements algorithmiques d'une des pièces centrales de l'informatique contemporaine: un moteur de rendu html. Concrètement, vous allez réaliser un moteur de rendu pour un langage html simplifié. Vous réaliserez ce moteur de rendu en python avec la bibliothèque pygame.

Comme tout document web, les pages html sont représentées par des arbres, la réalisation de votre moteur de rendu reposera donc sur des algorithmes d'arbres. Comme il s'agit d'un programme assez important, vous respecterez les bonnes pratiques de qualité de développement et notamment la méthodologie de tests. Vous travaillerez par groupe de trois personnes, il vous faudra donc utiliser `git` et suivre les règles du développement collaboratif.

Votre travail sera évalué sur les critères suivants:

- le bon fonctionnement de votre code
- l'efficacité des algorithmes que vous implémenterez
- la couverture de votre code par des tests pertinents
- votre utilisation de `git`
- la présentation de votre code, dont la documentation de l'utilisation et du code
- les explications de votre travail dans un bref rapport.

Vous aurez à faire un rendu partiel (sous forme de *tag* dans votre dépôt git) la semaine du 10 octobre, puis un rendu final la semaine du 28 novembre.

2 La collaboration avec Github

Dans ce projet, vous travailler dans une équipe de 3 personnes, sur un dépôt commun dans Github. Pour créer ou joindre votre équipe, il vous faut cliquer sur ce lien et suivre les instructions.

Vous devez créer un compte GitHub (si vous ne l'avez pas encore) et utiliser le vrai nom prénom car ce projet est évalué directement dans votre dépôts sur Github. Github ne permet pas d'utiliser un mot de passe pour travailler avec les dépôts, vous devrez créer des jetons d'accès personnels. Veuillez suivre les instructions du fichier **personal_access_token.pdf**

3 Représentation des documents mini-html

3.1 Le langage mini-html

L'enjeu de cette SAÉ est de vous initier aux principes *fondamentaux* du rendu de pages web. Lors de la consultation d'une page web, on compte trois étapes principales:

le chargement une connexion réseau est établie pour récupérer les données de la page web;

le parsing les données téléchargées sont lues pour créer un modèle abstrait du document, le DOM;

le rendu à partir du DOM, le navigateur crée un affichage interactif du contenu.

Les deux premières étapes, le chargement et le parsing sont faites pour vous. Le chargement est fait pour vous puisque l'on va récupérer des fichiers dans le répertoire courant. Le parsing sera fait par le code qui vous est donné. Il ne vous reste donc plus qu'à utiliser le DOM obtenu pour réaliser un affichage avec `pygame`.

Le langage `html` un nombre sans cesse croissant de balises. De plus, toute page web digne de ce nom compte, en plus du code html, du css et du javascript. Cela donne des documents d'une grande richesse, mais aussi d'une grande complexité. Pour rendre la tâche abordable, les documents que vous aurez à afficher seront donné en `mini-html`, un html *très* simplifié.

Voici la liste des balises de `mini-html`; elles ont sauf exception le même rôle que la balise correspondante en html:

- `<html>`
- `<head>` peut contenir un élément `<title>` qui donne le titre du document,
- `<body>`
- `<p>`
- `<div>`
- `<section>`
- `<heading>` correspond au titre d'une section, c'est à dire à un élément `h1` à `h6` en html. Son niveau est calculé à partir du niveau de la section à laquelle il appartient
- `<image>`
- `<rectangle/>` est un élément qui n'existe pas en html. Il s'agit d'un élément sans enfants dont les dimensions (`width=""` et `width=""`) sont fixées et données en attribut. Les documents dont toutes les feuilles sont des `<rectangle/>` vous seront particulièrement utiles pour tester et déboguer votre code de rendu.

Le code qui vous est donné contient une classe pour chaque type d'élément (notamment `class Document`, `class Para`, etc). La fonction `parsing.parse(chemin)` lit le fichier dont le chemin est passé en argument et construit une instance de la classe `Div` qui représente le document tout entier, ainsi qu'une instance de `Header` pour son en-tête.

Chacune des classes représentant éléments a les attributs suivants:

- `enfants`: une liste de ses enfants,

- `attrs`: un dictionnaire représentant ses attributs (par exemple `href, class, id` etc.)

Il vous est vivement conseillé d'implémenter des méthodes telles que:

- `balise()` qui renvoie le nom de la balise associé à l'élément
- `classes()` qui renvoie la liste des classes css de l'élément
- `identifiant()` qui renvoie la liste de tous les attributs de l'élément.
- `dimensions()` qui renvoie un couple représentant les dimensions du rectangle contenant l'élément.

4 Les tâches à accomplir

Le squelette qui vous est donné est un programme `browser.py`. Quand on appelle ce programme par `python3 browser.py ma_page.mini_html`, on obtient une fenêtre avec un texte bouche-trou. Votre travail est de modifier le code qui vous est donné pour obtenir un affichage correct pour des pages mini-html de plus en plus complexes. Des pages mini-html d'exemples vous sont données sur Célène par ordre de complexité croissante: la page `00_base.mini-html` est beaucoup plus abordable que la page `99_défi.mini-html`. On vous donne pour chacune une copie d'écran du document.

4.1 Les classes `Navigation` et `Contexte`

Dans le code qui vous est fourni (fichiers `browser.py` et `contexte.py`), vous trouverez deux classes auxiliaires `Navigation` et `Contexte`.

La classe `Contexte` contient les données auxiliaires qui sont nécessaires pour votre application pygame, notamment les polices. La classe `Navigation` représente l'état de la navigation, elle permettra d'afficher la classe courante et de passer d'une page à une autre quand vous implémenterez les liens (méthode `goto`).

4.2 Requêtes sur le document

Dans un premier temps, vous allez implémenter un programme qui affiche dans le terminal quelques données simples à partir du document mini-html qui lui est donné en argument. Ces requêtes seront:

- le nombre de paragraphes dans le document
- la liste des images qui apparaissent
- une table des matières reprenant les éléments *header* des *section*s du document.
- afficher l'élément avec un identifiant donné
- afficher tous les éléments qui ont une classe donnée.

Vous pourrez tester ces requêtes sur la page du fennec (52_fennec.mini_html).

4.3 Le modèle de boîte

Le moteur de rendu a proprement parler repose sur le modèle de boîtes du web. Les éléments de base sont représentés par des instances des classes `Para`, `Lien`, `Heading` ou `Imgage`, définies dans `mini_html.py`.

Pour afficher une page, il faut afficher chacun de ses nœuds. Pour cela, il faut créer un élément graphique pour chaque nœud de l'arbre. Pour pouvoir disposer ces éléments les uns par rapport aux autres, il faut également connaitre leurs dimensions. Pour chaque élément, il faut compter dans ses dimensions la place occupée par ses enfants.

4.4 Style

Il est possible de donner un style aux éléments mini-html de deux façons. Soit via des feuilles de style css, soit «à l'ancienne», directement dans le mini-html. Dans un premier temps, vous implémenterez le cas où le style est donné directement dans le fichier mini-html: même si cette solution est bien moins élégante, elle est plus simple à aborder dans un premier temps.

Les éléments peuvent avoir des attributs de style: `couleur_texte` et `couleur_fond` qui correspondent à la couleur du texte et à la couleur d'arrière-plan. Lorsque l'on spécifie un style pour un conteneur (div ou section), celui-ci est hérité par les éléments qu'il contient.

L'attribut `taille_police` modifie la taille de la police à utiliser pour afficher du texte. Si l'attribut `taille_police` d'un élément vaut +2, on utilisera une police plus grande de 2 tailles pour afficher l'élément. Quand plusieurs éléments `div` imbriqués modifient `taille_police`, les modifications sont cumulatives.

Les éléments `div` peuvent avoir un attribut `orientation`, qui peut prendre les valeurs `horizontal` ou `vertical`. Cet attribut détermine dans quelle direction les enfants du div vont s'agencer. Par défaut, ou si l'attribut est à `vertical`, les enfants se placent les uns en-dessous des autres. Si l'attribut est à `horizontal`, les enfants se placent de gauche à droite. Quand un élément `div` n'a pas d'attribut `orientation`, il doit hériter celui-ci de son parent.

Enfin, comme en html, les éléments peuvent avoir une `margin`.

5 Autres tâches sur la qualité dev

5.1 Assurer le code propre et la documentation

Le code est propre s'il peut être compris facilement - par tous les membres de l'équipe et par l'enseignant. Avec la compréhensibilité vient la lisibilité, l'extensibilité, la maintenabilité, la documentation (y compris des docstrings, des commentaires, et des annotations). Tout ce qu'il faut pour faire durer un projet dans la durée sans accumuler une grosse dette technique.

Cela leur facilitera grandement la collaboration si la disposition du code, la documentation, la convention de nommage, etc. sont identiques dans tous les fichiers.

5.2 Développement piloté par les tests

Il est recommandé d'écrire un test unitaire d'échec avant d'implémenter des fonctionnalités.

Le fichier tests.py vous est donné avec un exemple test unitaire. Vous devez écrire d'autres tests unitaires pour couvrir tout votre code. La couverture par des tests est obligatoire 100%.

Vous devez utiliser les tests pendant votre développement pour assurer le bon fonctionnement avant commiter et envoyer le code au dépot distant. Pour nous aider d'évaluer votre travail. Votre dépot Github est configuré pour :

- lancer ces tests automatiquement pour tous les "push" au dépôt distant.
- lancer le calcul de couverture de vos tests.

Un Makefile vous est fourni pour accéder rapidement aux commandes utiles : pylint, mypy, yapf, tests unitaires, vérification de la couverture. Explorez ce fichier pour mieux comprendre et pour faciliter votre travail.

6 Rendu intermédiaire

6.1 Consignes

Nous vous offrons deux "jours de retard" gratuits, utilisable pour les 2 deadlines. Soit utilisé pour ce rendu intermédiaire, soit utilisé dans le rendu final. Tous les autres retards sont pénalisés de 10% par jour.

Il vous faut créer un Tag et envoyer au dépôt distant:

- git tag -a v1.0 -m "Rendu intermédiaire"
- git push v1.0

6.2 Contenu du premier rendu

Le premier rendu consistera en deux programmes distincts.

Le premier, `analyse_mini_html.py` permettra d'afficher les différentes requêtes définies à la partie 4.2. La commande `python3 analyse_mini_html.py <requête> <fichier>` lancera la requête demandée sur le fichier passé en argument. Les valeurs possibles de `requête` sont:

- `nombre_par` qui compte les paragraphes dans le fichier
- `liste_images` qui fait la liste des images
- `table_matieres` qui affiche la table des matières

Le second programme, `rendu.py` sera le début de votre moteur de rendu. Il vous est demandé pour cette première version de pouvoir afficher correctement les exemples dont le numéro est inférieur à 50

7 Rendu final

7.1 Consignes

Nous vous offrons deux "jours de retard" gratuits, utilisable pour les 2 deadlines. Soit utilisé pour le rendu intémédiaire, soit utilisé dans ce rendu final. Tous les autres retards sont pénalisés de 10% par jour.

Il vous faut créer un Tag et envoyer au dépot distant:

- git tag -a v2.0 -m "Rendu final"
- git push v2.0

7.2 Contenu du rendu final

TBA