

Un livre de Wikilivres.

Vous avez certainement déjà appris par ailleurs un grand nombre de choses concernant la rédaction de pages web. Vous savez que ces pages sont des documents au format HTML, que l'on peut consulter *via* un réseau (intranet ou internet) à l'aide d'un logiciel appelé browser web ou *navigateur* (ex : *Firefox*, *Google Chrome*, *Konqueror*, *Internet Explorer*, ...).

Les pages HTML sont installées dans les répertoires publics d'un autre ordinateur où fonctionne en permanence un logiciel appelé *serveur Web* (*Apache*, *IIS*, *Zope*, ...). Lorsqu'une connexion a été établie entre cet ordinateur et le vôtre, votre logiciel navigateur peut dialoguer avec le logiciel serveur (par l'intermédiaire de toute une série de dispositifs matériels et logiciels dont nous ne traiterons pas ici : lignes téléphoniques, routeurs, caches, protocoles de communication ...).

Le *protocole HTTP* qui gère la transmission des pages web autorise l'échange de données dans les deux sens. Mais dans la grande majorité des cas, le transfert d'informations n'a pratiquement lieu que dans un seul, à savoir du serveur vers le navigateur : des textes, des images, des fichiers divers lui sont expédiés en grand nombre (ce sont les pages consultées) ; en revanche, le navigateur n'envoie guère au serveur que de toutes petites quantités d'information : essentiellement les adresses URL des pages que l'internaute désire consulter.

Pages web interactives

Vous savez cependant qu'il existe des sites web où vous êtes invité à fournir vous-même des quantités d'information plus importantes : vos références personnelles pour l'inscription à un club ou la réservation d'une chambre d'hôtel, votre numéro de carte de crédit pour la commande d'un article sur un site de commerce électronique, votre avis ou vos suggestions, etc.

Dans un cas comme ceux-là, vous vous doutez bien que l'information transmise doit être prise en charge, du côté du serveur, par un programme spécifique. Il faut donc que les pages web destinées à accueillir cette information soient dotées d'un mécanisme assurant son transfert vers le logiciel destiné à le traiter. Il faudra également que ce logiciel puisse lui-même transmettre en retour une information au serveur, afin que celui-ci puisse présenter le résultat de l'opération à l'internaute, sous la forme d'une nouvelle page web.

Le but du présent chapitre est de vous expliquer comment vous pouvez vous servir de vos compétences de programmeur Python pour ajouter une telle interactivité à un site web, en y intégrant de véritables applications.

Remarque importante : Ce que nous allons expliquer dans les paragraphes qui suivent sera directement fonctionnel sur l'intranet de votre établissement scolaire ou de votre entreprise (à la condition toutefois que l'administrateur de cet intranet ait configuré son serveur de manière appropriée). En ce qui concerne l'internet, par contre, les choses sont un peu plus compliquées. Il va de soi que l'installation de logiciels sur un ordinateur serveur relié à l'internet ne peut se faire qu'avec l'accord de son propriétaire. Si un fournisseur d'accès à l'internet a mis à votre disposition un certain espace où vous êtes autorisé à installer des pages web « statiques » (c'est-à-dire de simples documents à consulter), cela ne signifie pas pour autant que vous pourrez y faire fonctionner des scripts Python. Pour que cela puisse marcher, vous devrez demander une autorisation et un certain nombre de renseignements à votre fournisseur d'accès. Il faudra en particulier lui demander si vous pouvez activer des scripts CGI écrits en Python à partir de vos pages, et dans quel(s) répertoire(s) vous pouvez les installer.

L'interface CGI

L'interface CGI (pour *Common Gateway Interface*) est un composant de la plupart des logiciels serveurs de pages web. Il s'agit d'une passerelle qui leur permet de communiquer avec d'autres logiciels tournant sur le même ordinateur. Avec CGI, vous pouvez écrire des scripts dans différents langages (*Perl*, *C*, *Tcl*, *PHP*, *Python* ...).

Plutôt que de limiter le web à des documents écrits à l'avance, CGI permet de générer des pages web sur le champ, en fonction des données que fournit l'internaute par l'intermédiaire de son logiciel de navigation. Vous pouvez utiliser les scripts CGI pour créer une large palette d'applications : des services d'inscription en ligne, des outils de recherche dans des bases de données, des instruments de sondage d'opinions, des jeux, etc.

L'apprentissage de la programmation CGI peut faire l'objet de manuels entiers. Dans cet ouvrage d'initiation, nous vous expliquerons seulement quelques principes de base, afin de vous faire comprendre, par comparaison, l'énorme avantage que présentent les modules serveurs d'applications spécialisés tels que *Karrigell*, *CherryPy* ou *Zope*, pour le programmeur désireux de développer un site web interactif.

Une interaction CGI rudimentaire

Notre premier exemple sera constitué d'une page web extrêmement simple. Nous n'y placerons qu'un seul élément d'interactivité, à savoir un unique bouton. Ce bouton servira à lancer l'exécution d'un petit programme que nous décrirons par après.

Remarque : pour la bonne compréhension de ce qui suit, nous supposerons que l'administrateur réseau de votre établissement scolaire ou de votre entreprise a configuré un serveur web d'intranet d'une manière telle que vous puissiez installer des pages HTML et des scripts Python dans un répertoire personnel.

Veuillez donc encoder le document HTML ci-dessous à l'aide d'un éditeur quelconque :

```
1 <HTML>
2 <HEAD><TITLE>Exercice avec Python</TITLE></HEAD>
3 <BODY>
4
5 <DIV ALIGN="center">
6 <IMG SRC="penguin.gif">
7 <H2>Page Web interactive</H2>
8 <P>Cette page est associée à un script Python</P>
9
10 <FORM ACTION="http://Serveur/cgi-bin/input_query.py" METHOD="post">
11 <INPUT TYPE="submit" NAME="send" VALUE="Exécuter le script">
12 </FORM>
13
14 </DIV></BODY></HTML>
```

Vous savez certainement déjà que les balises initiales `<HTML>`, `<HEAD>`, `<TITLE>`, `<BODY>`, ainsi que les balises finales correspondantes, sont communes à tous les documents HTML. Nous ne détaillerons donc pas leur rôle ici.

La balise `<DIV>` utilisée à la ligne 5 sert habituellement à diviser un document HTML en sections distinctes. Nous l'utilisons ici pour définir une section dans laquelle tous les éléments seront centrés (horizontalement) sur la page.

À la ligne 6, nous insérons une petite image.

La ligne 7 définit une ligne de texte comme étant un titre de 2e importance.

La ligne 8 est un paragraphe ordinaire.

Les lignes 10 à 12 contiennent le code important (pour ce qui nous occupe ici). Les balises `<FORM>` et `</FORM>` définissent en effet un formulaire, c'est-à-dire une portion de page Web susceptible de contenir divers *widgets* à l'aide desquels l'internaute pourra exercer une certaine activité : champs d'entrée, boutons, cases à cocher, boutons radio, etc.

La balise `FORM` doit contenir deux indications essentielles : l'*action* à accomplir lorsque le formulaire sera expédié (il s'agit en fait de fournir ici l'adresse URL du logiciel à invoquer pour traiter les données transmises), et la *méthode* à utiliser pour transmettre l'information (en ce qui nous concerne, ce sera toujours la méthode `post`).

Dans notre exemple, le logiciel que nous voulons invoquer est un script Python nommé `input_query.py` qui est situé dans un répertoire particulier du serveur d'intranet. Sur de nombreux serveurs, ce répertoire s'appelle souvent *cgi-bin*, par pure convention. Nous supposerons ici que l'administrateur de votre intranet scolaire vous autorise à installer vos scripts Python dans le même répertoire que celui où vous placez vos pages web personnelles.

Vous devrez donc modifier la ligne 10 de notre exemple, en remplaçant l'adresse `http://Serveur/cgi-bin`

`/input_query.py` par ce que votre professeur vous indiquera^[1].

La ligne 11 contient la balise qui définit un *widget* de type « bouton d'envoi » (balise `<INPUT TYPE="submit">`). Le texte qui doit apparaître sur le bouton est précisé par l'attribut `VALUE = "texte"`. L'indication `NAME` est facultative dans le cas présent. Elle mentionne le nom du widget lui-même (au cas où le logiciel destinataire en aurait besoin).

Lorsque vous aurez terminé l'encodage de ce document, sauvegardez-le dans le répertoire que l'on vous a attribué spécifiquement pour y placer vos pages, sous un nom quelconque, mais de préférence avec l'extension `.html` ou `.htm` (par exemple : `essai.html`).

Le script Python `input_query.py` est détaillé ci-dessous. Comme déjà signalé plus haut, vous pouvez installer ce script dans le même répertoire que votre document HTML initial :

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # Affichage d'un formulaire HTML simplifié :
4  print "Content-Type: text/html\n"
5  print """
6  <H3><FONT COLOR="Royal blue">
7  Page web produite par un script Python
8  </FONT></H3>
9
10 <FORM ACTION="print_result.py" METHOD="post">
11 <P>Veuillez entrer votre nom dans le champ ci-dessous, s.v.p. :</P>
12 <P><INPUT NAME="visiteur" SIZE=20 MAXLENGTH=20 TYPE="text"></P>
13 <P>Veuillez également me fournir une phrase quelconque :</P>
14 <TEXTAREA NAME="phrase" ROWS=2 COLS=50>Mississippi</TEXTAREA>
15 <P>J'utiliserai cette phrase pour établir un histogramme.</P>
16 <INPUT TYPE="submit" NAME="send" VALUE="Action">
17 </FORM>
18 """
```

Ce script ne fait rien d'autre que d'afficher une nouvelle page web, laquelle contient encore une fois un formulaire, mais celui-ci nettement plus élaboré que le précédent.

La première ligne est absolument nécessaire : elle indique à l'interface CGI qu'il faut lancer l'interpréteur Python pour pouvoir exécuter le script. La seconde ligne spécifie l'encodage du code source.

La ligne 4 est indispensable. Elle permet à l'interpréteur Python d'initialiser un véritable document HTML qui sera transmis au serveur web. Celui-ci pourra à son tour le réexpédier au logiciel navigateur de l'internaute, et celui-ci le verra donc s'afficher dans la fenêtre de navigation.

La suite est du pur code HTML, traité par Python comme une simple chaîne de caractères que l'on affiche à l'aide de l'instruction `print`. Pour pouvoir y insérer tout ce que nous voulons, y compris les sauts à la ligne, les apostrophes, les guillemets, etc., nous délimitons cette chaîne de caractères à l'aide de « triples guillemets » (Rappelons également ici que les sauts à la ligne sont complètement ignorés en HTML : nous pouvons donc en utiliser autant que nous voulons pour « aérer » notre code et le rendre plus lisible).

Un formulaire HTML pour l'acquisition des données

Analysons à présent le code HTML lui-même. Nous y trouvons essentiellement un nouveau formulaire, qui comporte plusieurs paragraphes, parmi lesquels on peut reconnaître quelques *widgets*. La ligne 10 indique le nom du script CGI auquel les données du formulaire seront transmises : il s'agira bien évidemment d'un autre script Python.

À la ligne 12, on trouve la définition d'un *widget* de type « champ d'entrée » (Balise `INPUT`, avec `TYPE="text"`). L'utilisateur est invité à y encoder son nom. Le paramètre `MAXLENGTH` définit une longueur maximale pour la chaîne de caractères qui sera entrée ici (20 caractères, en l'occurrence). Le paramètre `SIZE` définit la taille du champ tel qu'il doit apparaître à l'écran, et le paramètre `NAME` est le nom que nous choisissons pour la variable destinée à

mémoriser la chaîne de caractères attendue.

Un second champ d'entrée un peu différent est défini à la ligne 14 (balise `TEXTAREA`). Il s'agit d'un réceptacle plus vaste, destiné à accueillir des textes de plusieurs lignes. (Ce champ est automatiquement pourvu d'ascenseurs si le texte à insérer se révèle trop volumineux). Ses paramètres `ROWS` et `COLS` sont assez explicites. Entre les balises initiale et finale, on peut insérer un texte par défaut (`Mississippi` dans notre exemple).

Comme dans l'exemple précédent, la ligne 16 contient la définition du bouton qu'il faudra actionner pour transmettre les données au script CGI destinataire, lequel est décrit ci-après.

Un script CGI pour le traitement des données

Le mécanisme utilisé à l'intérieur d'un script CGI pour réceptionner les données transmises par un formulaire HTML est fort simple, comme vous pouvez l'analyser dans l'exemple ci-dessous :

```

1  #!/usr/bin/python
2  # Traitement des données transmises par un formulaire HTML
3
4  import cgi                                # Module d'interface avec le serveur web
5  form = cgi.FieldStorage()                 # Réception de la requête utilisateur :
6                                          # il s'agit d'une sorte de dictionnaire
7  if form.has_key("phrase"):                # La clé n'existera pas si le champ
8      text = form["phrase"].value           # correspondant est resté vide
9  else:
10     text = "*** le champ phrase était vide ! ***"
11
12  if form.has_key("visiteur"):              # La clé n'existera pas si le champ
13     nomv = form["visiteur"].value          # correspondant est resté vide
14  else:
15     nomv = "mais vous ne m'avez pas indiqué votre nom"
16
17  print "Content-Type: text/html\n"
18  print ""
19  <H3>Merci, %s !</H3>
20  <H4>La phrase que vous m'avez fournie était : </H4>
21  <H3><FONT Color="red"> %s </FONT></H3>"" % (nomv, text)
22
23  histogr = {}
24  for c in text:
25     histogr[c] = histogr.get(c, 0) + 1
26
27  liste = histogr.items()                  # conversion en une liste de tuples
28  liste.sort()                            # tri de la liste
29  print "<H4>Fréquence de chaque caractère dans la phrase :</H4>"
30  for c, f in liste:
31     print 'le caractère <B>%s</B> apparaît %s fois <BR>' % (c, f)

```

Les lignes 4 et 5 sont les plus importantes :

Le module `cgi` importé à la ligne 4 assure la connexion du script Python avec l'interface CGI, laquelle permet de dialoguer avec le serveur web.

À la ligne 5, la fonction `FieldStorage()` de ce module renvoie un objet qui contient l'ensemble des données transmises par le formulaire HTML. Nous plaçons cet objet, lequel est assez semblable à un dictionnaire classique, dans la variable `form`.

Par rapport à un véritable dictionnaire, l'objet placé dans `form` présente la différence essentielle qu'il faudra lui appliquer la méthode `value()` pour en extraire les données. Les autres méthodes applicables aux dictionnaires, telles la méthode `has_key()`, par exemple, peuvent être utilisées de la manière habituelle.

Une caractéristique importante de l'objet dictionnaire retourné par `FieldStorage()` est qu'il ne possédera aucune clé pour les champs laissés vides dans le formulaire HTML correspondant.

Dans notre exemple, le formulaire comporte deux champs d'entrée, auxquels nous avons associé les noms `visiteur` et `phrase`. Si ces champs ont effectivement été complétés par l'utilisateur, nous trouverons leurs contenus dans l'objet dictionnaire, aux index « `visiteur` » et « `phrase` ». Par contre, si l'un ou l'autre de ces champs n'a pas été complété, l'index correspondant n'existera tout simplement pas. Avant toute forme de traitement de valeurs, il est donc indispensable de s'assurer de la présence de chacun des index attendus, et c'est ce que nous faisons aux lignes 7 à 15.

Exercices

1. Pour vérifier ce qui précède, vous pouvez par exemple désactiver (en les transformant en commentaires) les lignes 7, 9, 10, 12, 14 & 15 du script. Si vous testez le fonctionnement de l'ensemble, vous constaterez que tout se passe bien si l'utilisateur complète effectivement les champs qui lui sont proposés. Si l'un des champs est laissé vide, par contre, une erreur se produit.

Solution

1. Réfléchissez !

Remarque : le script étant lancé par l'intermédiaire d'une page web, les messages d'erreur de Python ne seront pas affichés dans cette page, mais plutôt enregistrés dans le journal des événements du serveur web. Veuillez consulter l'administrateur de ce serveur pour savoir comment vous pouvez accéder à ce journal. De toute manière, attendez-vous à ce que la recherche des erreurs dans un script CGI soit plus ardue que dans une application ordinaire.

Le reste du script est assez classique.

- Aux lignes 17 à 21, nous ne faisons qu'afficher les données transmises par le formulaire. Veuillez noter que les variables `nomv` et `text` doivent exister au préalable, ce qui rend indispensables les lignes 9, 10, 14 et 15.
- Aux lignes 23, 24 et 25, nous nous servons d'un dictionnaire pour construire un histogramme simple, comme nous l'avons expliqué à la page.



À faire...

- À la ligne 27, nous convertissons le dictionnaire résultant en une liste de tuples, pour pouvoir trier celle-ci dans l'ordre alphabétique à la ligne 28.
- La boucle `for` des lignes 30 et 31 se passe de commentaires.

Un serveur web en pur Python !

Dans les pages précédentes, nous vous avons expliqué quelques rudiments de programmation CGI afin que vous puissiez mieux comprendre comment fonctionne une application web. Mais si vous voulez véritablement développer une telle application (par exemple un site web personnel doté d'une certaine interactivité), vous constaterez rapidement que l'interface CGI est un outil trop sommaire. Son utilisation telle quelle dans des scripts se révèle fort lourde, et il est donc préférable de faire appel à des outils plus élaborés.

L'intérêt pour le développement web est devenu très important, et il existe donc une forte demande pour des interfaces et des environnements de programmation bien adaptés à cette tâche. Or, même s'il ne peut pas prétendre à l'universalité de langages tels que C/C++, Python est déjà largement utilisé un peu partout dans le monde pour écrire des programmes très ambitieux, y compris dans le domaine des serveurs d'applications web. La robustesse et la facilité de mise en oeuvre du langage ont séduit de nombreux développeurs de talent, qui ont réalisé des outils de

développement web de très haut niveau. Plusieurs de ces applications peuvent vous intéresser si vous souhaitez réaliser vous-même des sites web interactifs de différents types.

Les produits existants sont pour la plupart des logiciels libres. Ils permettent de couvrir une large gamme de besoins, depuis le petit site personnel de quelques pages, jusqu'au gros site commercial collaboratif, capable de répondre à des milliers de requêtes journalières, et dont les différents secteurs sont gérés sans interférence par des personnes de compétences variées (infographistes, programmeurs, spécialistes de bases de données, etc.).

Le plus célèbre de ces produits est le logiciel *Zope*, déjà adopté par de grands organismes privés et publics pour le développement d'intranets et d'extranets collaboratifs. Il s'agit en fait d'un système serveur d'applications, très performant, sécurisé, presque entièrement écrit en Python, et que l'on peut administrer à distance à l'aide d'une simple interface web. Il ne nous est pas possible de décrire l'utilisation de Zope dans ces pages : le sujet est trop vaste, et un livre entier n'y suffirait pas. Sachez cependant que ce produit est parfaitement capable de gérer de très gros sites d'entreprise en offrant d'énormes avantages par rapport à des solutions classiques telles que *PHP* ou *Java*.

D'autres outils moins ambitieux mais tout aussi intéressants sont disponibles. Tout comme Zope, la plupart d'entre eux peuvent être téléchargés librement depuis l'internet. Le fait qu'ils soient écrits en Python assure en outre leur portabilité : vous pourrez donc les employer aussi bien sous Windows que sous Linux ou MacOS. Chacun d'eux peut être utilisé en conjonction avec un serveur web « classique » tel que Apache ou Xitami (c'est préférable si le site à réaliser est destiné à supporter une charge de connexions très importante), mais certains d'entre eux intègrent en outre leur propre serveur web, ce qui leur permet de fonctionner également de manière tout à fait autonome. Cette possibilité se révèle particulièrement intéressante au cours de la mise au point d'un site, car elle facilite la recherche des erreurs.

Cette totale autonomie alliée à la grande facilité de leur mise en oeuvre fait de ces produits de fort bonnes solutions pour la réalisation de sites web d'intranet spécialisés, notamment dans des petites et moyennes entreprises, des administrations, ou dans des écoles. Si vous souhaitez développer une application Python qui soit accessible par l'intermédiaire d'un simple navigateur web, via un intranet d'entreprise (ou même via l'internet, si la charge prévisible n'est pas trop importante), ces applications sont faites pour vous.

Il en existe une grande variété : *Poor man's Zope*, *Spyce*, *Karrigell*, *Webware*, *CherryPy*, *Quixote*, *Twisted*, etc. Choisissez en fonction de vos besoins : vous n'aurez que l'embarras du choix.

Dans les lignes qui suivent, nous allons décrire une petite application web fonctionnant à l'aide de *Karrigell*. Vous pouvez trouver ce système à l'adresse : <http://karrigell.sourceforge.net>. Il s'agit d'une solution de développement web simple, bien documentée en anglais et en français (son auteur, Pierre Quentel, est en effet originaire de Bretagne, tout comme le mot *karrigell*, d'ailleurs, lequel signifie « charrette »).

Installation de Karrigell

L'installation de Karrigell est un jeu d'enfant : il vous suffit d'extraire dans un répertoire quelconque le fichier archive que vous aurez téléchargé depuis l'internet. L'opération de désarchivage crée automatiquement un sous-répertoire nommé *Karrigell-numéro de version*. C'est ce répertoire que nous considérerons comme répertoire racine dans les lignes qui suivent.

Si vous ne comptez pas utiliser le serveur de bases de données Gadfly^[2] qui vous est fourni en complément de Karrigell lui-même, c'est tout ! Sinon, entrez dans le sous-répertoire gadfly-1.0.0 et lancez la commande : `python setup.py install` (Sous Linux, il faut être root). Vous devez effectuer cette opération si vous souhaitez visualiser la totalité de la démonstration intégrée.

Démarrage du serveur :

Il s'agit donc bel et bien de mettre en route un *serveur* web, auquel vous pourrez accéder ensuite à l'aide d'un navigateur quelconque, localement ou par l'intermédiaire d'un réseau. Avant de le faire démarrer, il est cependant conseillé de jeter un petit coup d'oeil dans son fichier de configuration, lequel se nomme *Karrigell.ini* et se trouve dans le répertoire-racine.

Par défaut, Karrigell attend les requêtes *http* sur le port n° 80. Et c'est bien ce numéro de port que la plupart des logiciels navigateurs utilisent eux-mêmes par défaut. Cependant, si vous installez Karrigell sur une machine Linux dont vous n'êtes pas l'administrateur, vous n'avez pas le droit d'utiliser les numéros de port inférieurs à 1024 (pour des raisons de sécurité). Si vous êtes dans ce cas, vous devez donc modifier le fichier de configuration afin que Karrigell utilise un numéro de port plus élevé. En général, vous choisirez d'enlever simplement le caractère # au début de la ligne 39, ce qui activera l'utilisation du n° de port 8080. Plus tard, vous souhaiterez peut-être encore modifier le fichier de configuration afin de modifier l'emplacement du répertoire racine pour votre site web (par défaut, c'est le répertoire du serveur lui-même).

Une fois le fichier de configuration modifié, entrez dans le répertoire racine du serveur, si vous n'y êtes pas déjà, et lancez simplement la commande :

```
python Karrigell.py
```

C'est tout. Votre serveur Karrigell se met en route, et vous pouvez en vérifier le fonctionnement tout de suite à l'aide de votre navigateur web préféré. Si vous lancez celui-ci sur la même machine que le serveur, vous le dirigerez vers une adresse telle que : *http://localhost:8080/index.html*, « localhost » étant le terme consacré pour désigner la machine locale, « 8080 » le numéro de port choisi dans le fichier de configuration, et « index.html » le nom du fichier qui contient la page d'accueil du site. Par contre, si vous voulez accéder à cette même page d'accueil depuis une autre machine, vous devrez (dans le navigateur de celle-ci) indiquer le nom ou l'adresse IP du serveur, en lieu et place de *localhost*.

Avec l'adresse indiquée au paragraphe précédent^[3], vous atteindrez la page d'accueil d'un site de démonstration de Karrigell, qui est déjà pré-installé dans le répertoire racine. Vous y retrouverez la documentation de base, ainsi que toute une série d'exemples.

Dans ce qui précède, il est sous-entendu que vous avez lancé le serveur depuis une console texte, ou depuis une fenêtre de terminal. Dans un cas comme dans l'autre, les messages de contrôle émis par le serveur apparaîtront dans cette console ou cette fenêtre. C'est là que vous pourrez rechercher des messages d'erreur éventuels. C'est là aussi que vous devrez intervenir si vous voulez arrêter le serveur (avec la combinaison de touches CTRL-C).

Ébauche de site web

Essayons à présent de réaliser notre propre ébauche de site web. À la différence d'un serveur web classique, Karrigell peut gérer non seulement des pages HTML statiques (fichiers *.htm*, *.html*, *.gif*, *.jpg*, *.css*) mais également :

- des scripts Python (fichiers *.py*) ;
- des scripts hybrides Python Inside HTML (fichiers *.pih*) ;
- des scripts hybrides HTML Inside Python (fichiers *.hip*).

Laissons de côté les scripts hybrides, dont vous pourrez étudier vous-même la syntaxe (par ailleurs très simple) si vous vous lancez dans une réalisation d'une certaine importance (ils pourront vous faciliter la vie). Dans le contexte limité de ces pages, nous nous contenterons de quelques expériences de base avec des scripts Python ordinaires.

Comme tous les autres éléments du site (fichiers *.html*, *.gif*, *.jpeg*, etc.), ces scripts Python devront être placés dans le répertoire racine^[4]. Vous pouvez tout de suite effectuer un test élémentaire en rédigeant un petit script d'une seule ligne, tel que :

```
print "Bienvenue sur mon site web !"
```

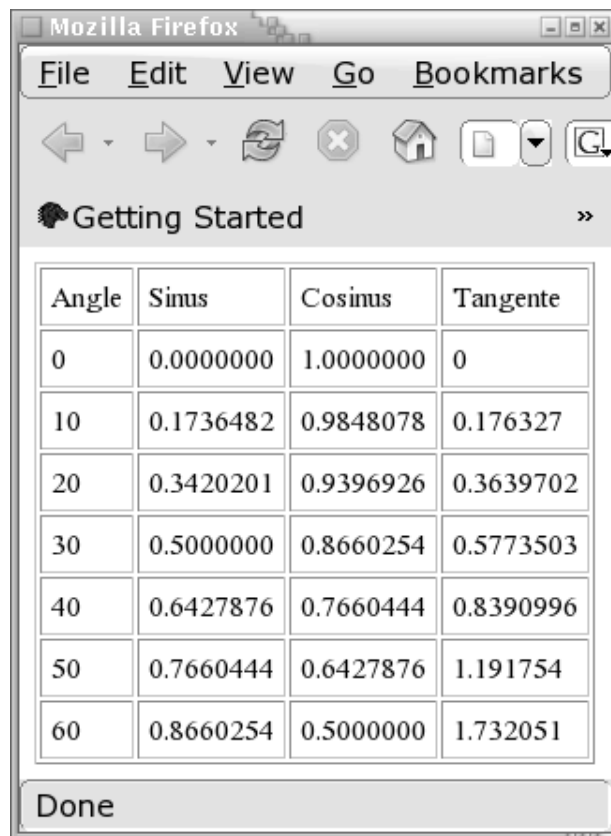
Sauvegardez ce script sous le nom *hello.py* dans le répertoire racine, puis entrez l'adresse : *http://localhost/hello.py* (ou même : *http://localhost/hello* - l'extension *.py* peut être omise) dans votre navigateur. Vous devriez y voir apparaître le message. Cela signifie donc que dans l'environnement Karrigell, la sortie de l'instruction *print* est redirigée vers la fenêtre du navigateur client, plutôt que la console (ou la fenêtre de terminal) du serveur.

Étant donné que l'affichage a lieu dans une fenêtre de navigateur web, vous pouvez utiliser toutes les ressources de la syntaxe HTML afin d'obtenir un formatage déterminé. Vous pouvez par exemple afficher un petit tableau de 2 lignes et 3 colonnes, avec les instructions suivantes :

```
print """
<TABLE BORDER="1" CELLPADDING="5">
<TR> <TD> Rouge </TD> <TD> Vert </TD> <TD> Bleu </TD> </TR>
<TR> <TD> 15 % </TD> <TD> 62 % </TD> <TD> 23 % </TD> </TR>
</TABLE>
"""
```

Rappelons que la balise `TABLE` définit un tableau. Son option `BORDER` spécifie la largeur des bordures de séparation, et `CELLPADDING` l'écart à réserver autour du contenu des cellules. Les Balises `TR` et `TD` (*Table Row* et *Table Data*) définissent les lignes et les cellules du tableau.

Vous pouvez bien entendu utiliser également toutes les ressources de Python, comme dans l'exemple ci-dessous où nous construisons une table des sinus, cosinus et tangentes des angles compris entre 0° et 90°, à l'aide d'une boucle classique.



Angle	Sinus	Cosinus	Tangente
0	0.0000000	1.0000000	0
10	0.1736482	0.9848078	0.176327
20	0.3420201	0.9396926	0.3639702
30	0.5000000	0.8660254	0.5773503
40	0.6427876	0.7660444	0.8390996
50	0.7660444	0.6427876	1.191754
60	0.8660254	0.5000000	1.732051

```
1 from math import sin, cos, tan, pi
2
3 # Construction de l'en-tête du tableau avec les titres de colonnes :
4 print """<TABLE BORDER="1" CELLPADDING="5">
5 <TR><TD>Angle</TD><TD>Sinus</TD><TD>Cosinus</TD><TD>Tangente</TD></TR>"""
6
7 for angle in range(0,62,10):
8     # conversion des degrés en radians :
9     aRad = angle * pi / 180
10    # construction d'une ligne de tableau, en exploitant le formatage des
11    # chaînes de caractères pour figurer l'affichage :
12    print "<TR><TD>%s</TD><TD>%8.7f</TD><TD>%8.7f</TD><TD>%8.7g</TD></TR>" \
13          % (angle, sin(aRad), cos(aRad), tan(aRad))
14
15 print "</TABLE>"
```


Commentaires

Ligne 7 : Nous nous servons de la fonction `range()` pour définir la gamme d'angles à couvrir (de zéro à 60 degrés par pas de 10).

Ligne 9 : Les fonctions trigonométriques de Python nécessitent que les angles soient exprimés en radians. Il faut donc effectuer une conversion.

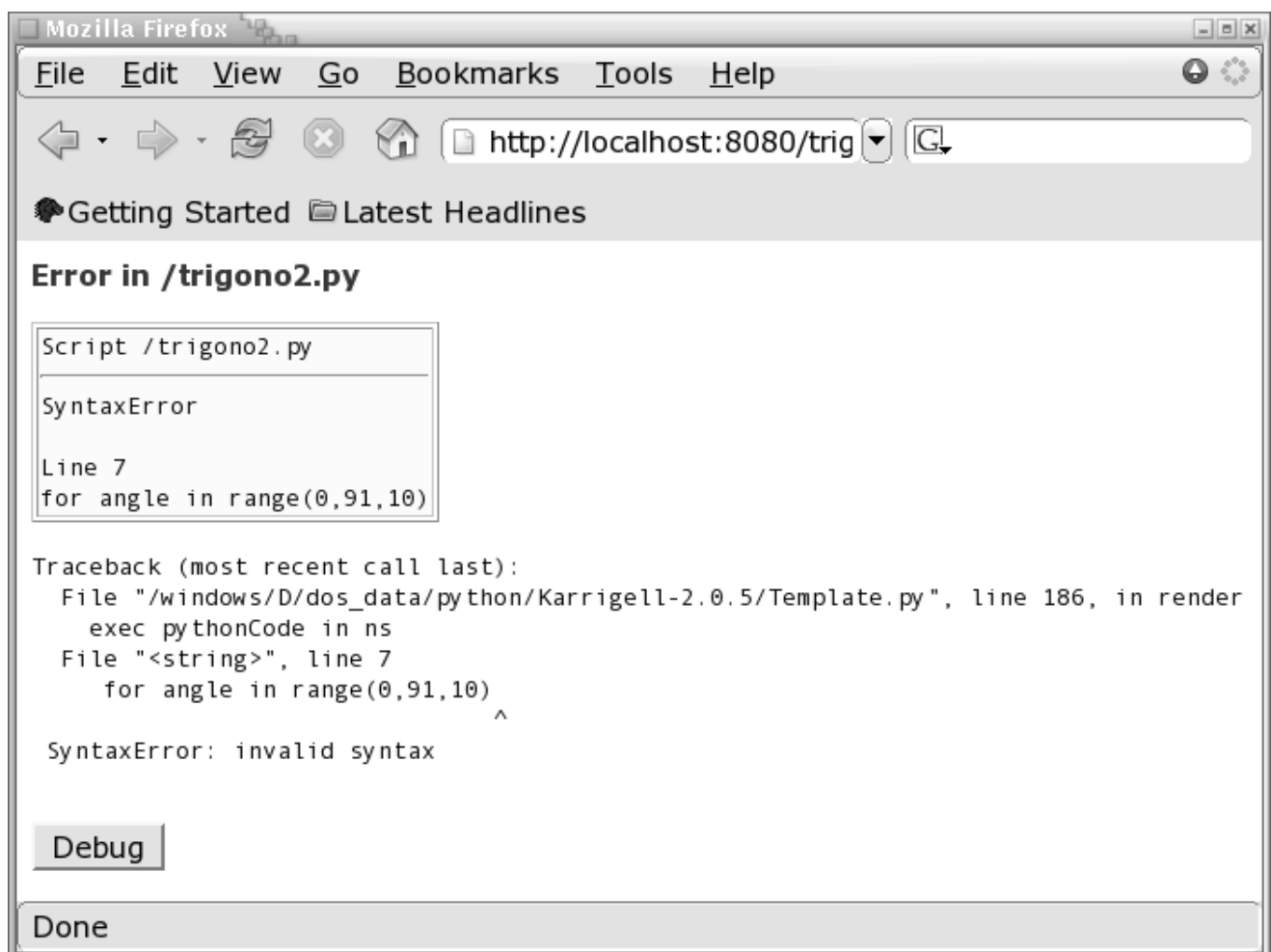
Ligne 12 : Chaque ligne du tableau comporte quatre valeurs, lesquelles sont mises en forme à l'aide du système de formatage des chaînes de caractères décrit déjà à la page

: le marqueur de conversion `%8.7f` force un affichage à 8 chiffres, dont 7 après la « virgule » décimale. Le marqueur `%8.7g` fait à peu près la même chose, mais passe à la notation scientifique lorsque c'est nécessaire. À ce stade, vous vous demandez peut-être où se situe la différence entre ce que nous venons d'expérimenter ici et un script CGI classique (tels ceux des pages

et suivantes).

L'intérêt de travailler dans un environnement plus spécifique tel que Karrigell apparaît cependant très vite si vous faites des erreurs. En programmation CGI classique, les messages d'erreur émis par l'interpréteur Python ne s'affichent pas dans la fenêtre du navigateur. Ils sont enregistrés dans un fichier journal du serveur (Apache, par exemple), ce qui ne facilite pas leur consultation.

Avec un outil comme Karrigell, par contre, vous disposez d'une signalisation très efficace, ainsi que d'un outil de débogage complet. Faites l'expérience d'introduire une petite erreur dans le script ci-dessus, et relancez votre navigateur sur la page modifiée. Par exemple, en supprimant le double point à la fin de la ligne 7, nous avons obtenu nous-mêmes l'affichage suivant :



En cliquant sur le bouton « Debug », on obtient encore une foule d'informations complémentaires (affichage du

script complet, variables d'environnement, etc.).

Prise en charge des sessions

Lorsque l'on élabore un site web interactif, on souhaite fréquemment que la personne visitant le site puisse s'identifier et fournir un certain nombre de renseignements tout au long de sa visite dans différentes pages (l'exemple type étant le remplissage d'un « caddy » au cours de la consultation d'un site commercial), toutes ces informations étant conservées quelque part jusqu'à la fin de sa visite. Et il faut bien entendu réaliser cela indépendamment pour chaque client connecté.

Il serait possible de transmettre les informations de page en page à l'aide de champs de formulaires cachés, mais ce serait compliqué et très contraignant. Il est préférable que le système serveur soit doté d'un mécanisme spécifique, qui attribue à chaque client une *session* particulière. Karrigell réalise cet objectif par l'intermédiaire de *cookies*. Lorsqu'un nouveau visiteur du site s'identifie, le serveur génère un *cookie* appelé *sessionId* et l'envoie au navigateur web, qui l'enregistre. Ce cookie contient un « identifiant de session » unique, auquel correspond un objet-session sur le serveur. Lorsque le visiteur parcourt les autres pages du site, son navigateur renvoie à chaque fois le contenu du cookie au serveur, et celui-ci peut donc retrouver l'objet-session correspondant, à l'aide de son identifiant. L'objet-session reste donc disponible tout au long de la visite de l'internaute : il s'agit d'un objet Python ordinaire, dans lequel on mémorise un nombre quelconque d'informations sous forme d'attributs.

Au niveau de la programmation, voici comment cela se passe :

Pour chaque page dans laquelle vous voulez consulter ou modifier une information de session, vous commencez par créer un objet de la classe `Session()` :

```
objet_session = Session()
```

Si vous êtes au début de la session, Karrigell génère un identifiant unique, le place dans un cookie et envoie celui-ci au navigateur web. Vous pouvez alors ajouter un nombre quelconque d'attributs à l'objet-session :

```
objet_session.nom = "Jean Dupont"
```

Dans les autres pages, vous procédez de la même manière, mais l'objet produit dans ce cas par la classe `Session()` n'est pas nouveau : c'est l'objet créé en début de session, retrouvé en interne par le serveur grâce à son identifiant relu dans le cookie. Vous pouvez accéder aux valeurs de ses attributs, et aussi en ajouter de nouveaux :

```
obj_sess = Session()           # récupérer l'objet indiqué par le cookie
nom = obj_sess.nom              # retrouver la valeur d'un attribut existant
obj_sess.article = 49137        # ajouter un nouvel attribut
```

Les objets-sessions prennent aussi en charge une méthode `close()`, qui a pour effet d'effacer l'information de session. Vous n'êtes cependant pas obligé de clore explicitement les sessions : Karrigell s'assure de toute façon qu'il n'y ait jamais plus de 1000 sessions simultanées : il efface les plus anciennes quand on arrive à la 1000ème.

Exemple de mise en oeuvre

Sauvegardez les trois petits scripts ci-dessous dans le répertoire-racine. Le premier génère un formulaire HTML similaire à ceux qui ont été décrits plus haut. Nommez-le *sessionTest1.py* :

```
1 # Affichage d'un formulaire d'inscription :
2
3 print ""
4 <H3>Veuillez vous identifier, SVP </H3>
5
```

```

6 <FORM ACTION = "sessionTest2.py">
7 Votre nom : <INPUT NAME = "nomClient"> <BR>
8 Votre prénom : <INPUT NAME = "prenomClient"> <BR>
9 Votre sexe (m/f) : <INPUT NAME = "sexeClient" SIZE = "1"> <BR>
10 <INPUT TYPE = "submit" VALUE = "OK">
11 </FORM>"""

```

Le suivant sera nommé *sessionTest2.py*. C'est le script mentionné dans la balise d'ouverture du formulaire ci-dessus à la ligne 6, et qui sera invoqué lorsque l'utilisateur actionnera le bouton mis en place à la ligne 10. Ce script recevra les valeurs entrées par l'utilisateur dans les différents champs du formulaire, par l'intermédiaire d'un dictionnaire de requête situé dans la variable d'environnement `QUERY` de Karrigell^[5] :

```

1 obSess = Session()
2
3 obSess.nom = QUERY["nomClient"]
4 obSess.prenom = QUERY["prenomClient"]
5 obSess.sexe = QUERY["sexeClient"]
6
7 if obSess.sexe.upper() == "M":
8     vedette = "Monsieur"
9 else:
10    vedette = "Madame"
11 print "<H3> Bienvenue, %s %s </H3>" % (vedette, obSess.nom)
12 print "<HR>"
13 print ""
14 <a href = "sessionTest3.py"> Suite...</a>"""

```

La première ligne de ce script crée l'objet-session, génère pour lui un identifiant unique, et expédie celui-ci au navigateur sous la forme d'un cookie.

Dans les lignes 3, 4, 5, on récupère les valeurs entrées dans les champs du formulaire précédent, en utilisant leurs noms comme clés d'accès au dictionnaire de requêtes.

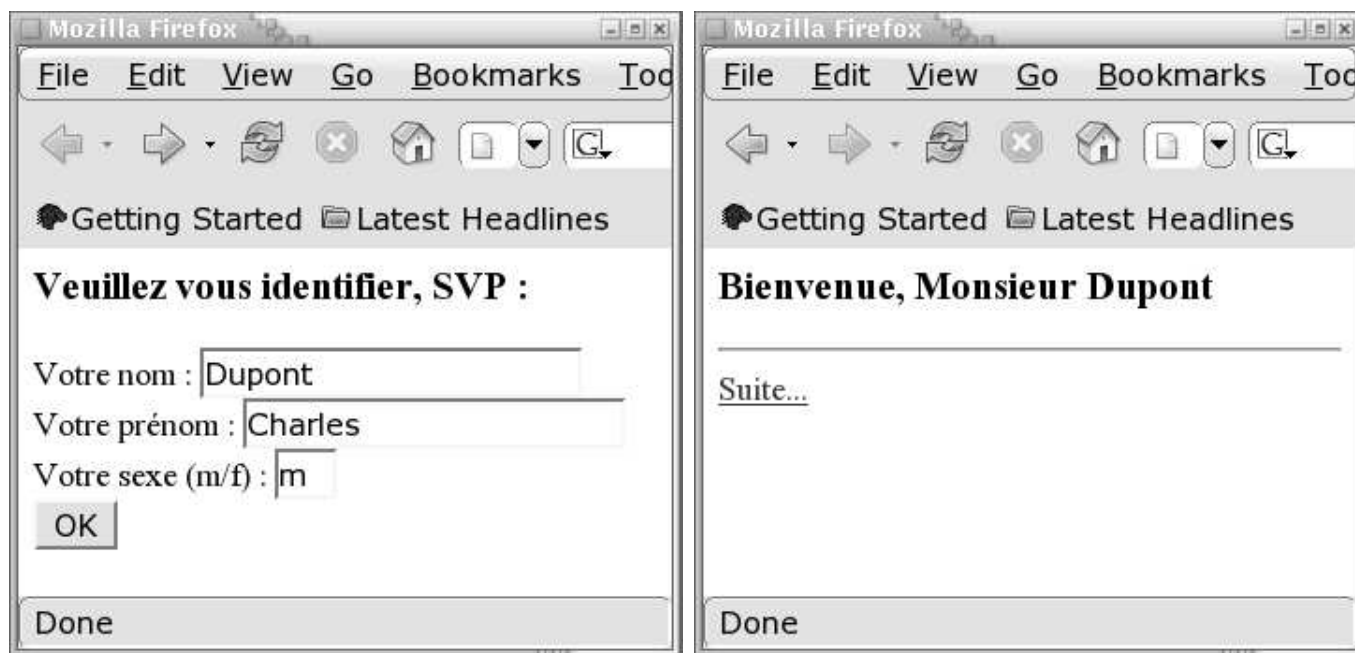
La ligne 14 définit un lien http pointant vers le troisième script, nommé *sessionTest3.py* :

```

1 suiviSess = Session()           # retrouver l'objet-session
2 suiviSess.article = 12345       # lui ajouter des attributs
3 suiviSess.prix = 43.67
4
5 print ""
6 <H3> Page suivante </H3> <HR>
7 Suivi de la commande du client : <BR> %s %s <BR>
8 Article n° %s, Prix : %s <HR>
9 "" % (suiviSess.prenom, suiviSess.nom,
10      suiviSess.article, suiviSess.prix)

```

Dirigez votre navigateur web vers l'adresse : `http://localhost:8080/sessionTest1`. Entrez des valeurs de votre choix dans les champs du formulaire, et cliquez sur le bouton OK :



Comme attendu, les informations entrées dans le formulaire sont transmises à la deuxième page. À présent, si vous cliquez sur le lien : « Suite... » dans celle-ci, vous dirigez encore une fois votre navigateur vers une nouvelle page, mais celle-ci n'aura fait l'objet d'aucune transmission de données (puisque l'on n'y accède pas par l'intermédiaire d'un formulaire). Dans le script *sessionTest3.py* qui génère cette page, vous ne pouvez donc pas utiliser la variable `QUERY` pour retrouver les informations entrées par le visiteur.

C'est ici qu'intervient le mécanisme des objets-sessions. Lors du lancement de ce troisième script, le cookie mémorisé par le navigateur est relu par le serveur, ce qui lui permet de régénérer l'objet-session créé dans le script précédent.



Analysez les trois premières lignes du script *sessionTest3.py* : l'objet `suiviSess` instancié à partir de la classe `Session()` est l'objet-session régénéré. Il contient les informations sauvegardées à la page précédente, et on peut lui en ajouter d'autres dans des attributs supplémentaires.

Vous aurez compris que vous pouvez désormais récupérer toutes ces informations de la même manière dans n'importe quelle autre page, car elles persisteront jusqu'à ce que l'utilisateur termine sa visite du site, à moins que vous ne fermiez vous-même cette session par programme, à l'aide de la méthode `close()` évoquée plus haut.

Exercices

1. Ajoutez au script précédent un lien vers une quatrième page, et écrivez le script qui générera celle-ci. Les informations devront cette fois être affichées dans un tableau :

Nom	Prénom	Sexe	Article	Prix
-----	--------	------	---------	------

Solution

1. Réfléchissez !

Autres développements

Nous terminons ici cette brève étude de Karrigell, car il nous semble vous avoir expliqué l'essentiel de ce qu'il vous faut connaître pour démarrer. Si vous désirez en savoir davantage, il vous suffira de consulter la documentation et les exemples fournis avec le produit. Comme nous l'avons déjà signalé plus haut, l'installation de Karrigell inclut l'installation du système de bases de données Gadfly. Vous pouvez donc très rapidement et très aisément réaliser un site interactif permettant la consultation à distance d'un ensemble de données quelconques, en admettant bien entendu que la charge de requêtes de votre site reste modérée, et que la taille de la base de données elle-même ne devienne pas gigantesque. N'espérez pas gérer à l'aide de Karrigell un site commercial susceptible de traiter plusieurs millions de requêtes journalières !

Si vous ambitionnez de réaliser ce genre de choses, il vous faudra étudier d'autres offres logicielles, comme par exemple *CherryPy* ou *Zope* associés à *Apache* pour le système serveur, et *SQLite*, *MySQL* ou *PostgreSQL* pour le gestionnaire de bases de données.

Notes

1. Par exemple : http://192.168.0.100/cgi/Classe6A/Dupont/input_query.py
2. Voyez le chapitre précédent : Gadfly est un serveur de bases de données écrit en Python.
3. Si vous avez laissé en place le n° de port par défaut (80), il est inutile de le rappeler dans les adresses, puisque c'est ce n° de port qui est utilisé par défaut par la plupart des navigateurs. Une autre convention consiste à considérer que la page d'accueil d'un site Web se trouve presque toujours dans un fichier nommé *index.htm* ou *index.html*. Lorsque l'on souhaite visiter un site Web en commençant par sa page d'accueil, on peut donc en général omettre ce nom dans l'adresse. Karrigell respecte cette convention, et vous pouvez donc vous connecter en utilisant une adresse simplifiée telle que : *http://localhost:8080* ou même : *http://localhost* (si le n° de port est 80).
4. ...ou bien dans des sous-répertoires du répertoire racine, comme il est d'usage de le faire lorsque l'on cherche à structurer convenablement le site en construction. Il vous suffira dans ce cas d'inclure le nom de ces sous-répertoires dans les adresses correspondantes.
5. Karrigell met en place un certain nombre de variables globales dont les noms sont en majuscules pour éviter un conflit éventuel avec les vôtres. Celle-ci joue le même rôle que la fonction `FieldStorage()` du module *cgi*. Veuillez consulter la documentation de Karrigell si vous souhaitez obtenir des explications plus détaillées.

Récupérée de « https://fr.wikibooks.org/w/index.php?title=Apprendre_à_programmer_avec_Python/Applications_web&oldid=531780 »

Dernière modification de cette page le 11 novembre 2016, à 00:35.

Les textes sont disponibles sous licence Creative Commons attribution partage à l'identique ; d'autres termes peuvent s'appliquer.

Voyez les termes d'utilisation pour plus de détails.