Python による HTML ファイル作成の省力化について

0448011

目　次

――Python による HTML ファイル作成の省力化について――

1　序論

(1) はじめに

福田洋一ゼミの基本的なテーマは「人の役に立つものを作る」である。このことを踏まえ、私が卒業製作に選んだテーマは「Python [(1)]による HTML ファイル作成の省力化について」である。HTML [(2)]をタグ[(3)]で記述する際に必要となるタグを閉じるというような単純な作業をプログラムに任せ、入力の省力化をするというのが目的である。

(2) 動機と現状

われわれのゼミでは 3 回生のとき、自分たち人文情報学科の学科ホームページを HTML に慣れるために製作する。製作はすべてテキストエディター[(4)]を使い、HTML タグを打ち込むことで進められた。この作業を進めていく際に終了の HTML タグ[(5)]を打ち込むことに面倒臭さを覚えた。この単純な「閉じる」という作業は人力でないと処理出来ないのかという疑問が出てきたからである。また、ホームページ製作の最終工程では HTML の文法チェックを行なったが、減点のいくつかは終了タグによるミスであった。もし、この作業を自動化できればミスを減らすことが出来るはずである。

調べてみたところ、HTML を作成するツールはいくつも存在するが、ホームページをイラストのようにレイアウトして作成できるようなものであったり、タグ入力を補佐してくれるテキストエディターであった。これらの問題点としてレイアウトするタイプのものは製作者が意図しないタグを使いレイアウトを整えることがあり、ソースコード[(6)]を後でみても可読性が著しく低くなることがあり、継続して HTML を編集して

いくには不向きである。タグ入力を補佐するテキストエディターでは確かに補佐はしてくれるが閉じタグを自動で入力してくれるようなシステムのものではない。また使用するテキストエディターが限定されるという弊害もある。

そこでテキストエディターに依存せず、タグを補完してくれるものを製作することを決めた。

(3) 対象者

製作するものは Python を使い変換作業をする必要があるので Python の知識を対象者は持つことが望ましい。また、HTML をタグで打つことから HTML の知識にも優れていることが望ましい。以上を踏まえた上で、HTML をタグで製作する必要のある人で省力化を望む人が対象者である。

2 製作過程

(1) 製作方針

まず初めに、どのように挙動するものが欲しいのかを明確にした。

1. 閉じタグを入力する必要のない HTML の作成
2. HTML のソースコード構造が見た目で分かる
3. テキストエディターを選ばない

この 3 つの条件をかなえ、かつ自動化できる部分は出来る限りプログラムに任せることで省力化出来るものを製作することにした。

また、HTML はいくつもバージョンがあるが 4.01 準拠とすることに決めた。

この条件 1 の理由として、閉じタグを自動で入力することにより入力の手間を省くことが出来、かつタグを忘れる人為的ミスを減らすことが出来るためである。条件 2 の理由として HTML ソースコードの構造を分かりやすくすることで後からでも加筆、修正を容易にするためである。条件 3 の理由は HTML をタグから作成する多くの人はテキストエディターをすでに使っている可能性が高く、乗り換えをせずに使用できるものが良いと考えるからである。また HTML バージョンを 4.01 準拠[(7)]にした理由は現時点で最新の表記ルールに則ることでソースコードの有用性を落とさないためである。加えて HTML は表記の自由度が高く、プログラムするにあたり規則性の強いものが求められた。そこで表記の規則性が比較的高い HTML4.01 準拠とした。

(2) 製作にあたって

プログラムを作るにあたっては Python を使用することにした。理由として HTML の構造が見ためで分かるという条件に Python のプログラム表記ルールであるインデントでプログラムのブロックを表現する方法がマッチするためである。また福田洋一先生が Python において HTML を表記するモジュール[(8)]を作っておられたからでもある。

福田先生の作成したモジュールは登録された HTML タグが極端に少なく、またタグに必要な属性[(9)]を書くことも出来なかった。このモジュールを拡張し、問題をクリアすることで目標とする二つの条件が満たされると推測し、製作を決めた。

モジュールを作る前にまず、HTML として何のタグが必要なのか、どのような表記が HTML4.01 準拠なのかを調べた。

HTML を調べるにあたっては『詳解　HTML & XHTML & CSS

辞典　第三版』を使い、HTML4.01 準拠のみを収録することとした。Python を学ぶにあたっては『みんなの Python』、『はじめての Python』を使用した。

(3) 動作環境

製作は Windows のサクラエディタを使用した。また作成するモジュールは単体で動くものではなく Python[10]上で動くものである。よって Windows で使用する場合、あらかじめ Python をインストールする必要がある。この場合、コマンドプロンプトで Python を起動するため Windows スタートボタンより設定を変更し、システム内の詳細設定から環境変数を選択し、システム環境変数の PATH を編集し、項目の最後に「;C:\Python25」と付け加える操作を行う。モジュールは実行したいファイルと同じフォルダに入れて準備は整う。

ここまで行った上で、コマンドプロンプトを開き、コマンドライン[11]で実行したいファイル名のあと半角スペースを空けて「>」を入力し、この後にまた半角スペースを空けて作成したいファイル名を指定し、Python を実行する。**（例：** `z:\>python test.py > test.html`**）**

Enter キーを押して、エラーがなければ（`z:\>`が返ってくる）変換は成功している。変換に成功した場合、実行したファイルと同じフォルダに作成したファイルで保存されている。しかし、なんらかのエラーがあれば、ファイル名の綴りミスや作成したファイルの文法ミスなどを修正し、再度実行する。

3　省力化モジュールの製作

(1) タグの要素処理に関する基本構成

タグの要素の入力に関して大まかに分けて、二つ処理するべき項目がある。タグに囲まれた本文となる部分とタグの属性である。タグに何を属性として追加するかは使用者の考えなので、必要な場合のみ書き加えられるようにすることでシンプルで見やすいものを作成する必要がある。

Python でこの構造を再現するためには「タグ要素を作る命令」、続いて「タグに囲まれる文章」、そして「必要があれば属性を入力する」三つが必要となる。これらを処理するためにはタグの要素を処理する関数を定義する必要がでてくる。

(i) 関数の引数処理

関数の引数は代入によって渡される。Python には関数の引数は位置による対応によって渡す方法と引数名を指定し渡す方法がある。位置による対応だけですべての引数を処理しようとすると引数名と同じ数の引数を同じ順番で渡さなければならない。引数名を指定する方法は関数の引数名にデフォルト値を設定し、渡された引数に該当するものがない場合にはデフォルト値で処理するようにすることが出来る。

この機能を使い、「タグに囲まれる文章」を位置による引数として処理し、属性の入力が必要な場合は引数名を指定し、属性に関する情報を入力することが出来る。属性の引数に関しては、関数内部に if 条件を設定し、もし属性に関する引数が入力されたならば、タグに属性を追加する処理を行い、属性に関する引数が入力されていないならばその属性は追加しないというようにした。具体的には属性の引数のデフォルトの値

を空文字列に指定し、属性の引数が空文字列以外の場合は属性入力がされているという条件を作成した。

しかし、この方法では一部のタグに手間が増えることになる。例を挙げると<img>タグにはタグ自身に囲まれる文章というものは存在しないが、入力すべき属性が最低でも二つ（属性 src、属性 alt）存在する。このような場合においては必須の属性については属性を呼び出す名前をつけるのではなく、引数の位置によって引き渡すようにした。この方法だと属性の変数名を入力する必要がなくなり、入力を効率よく出来るためである。だが、使用するタグ定義名によって必要とする引数の数が違うことは入力ミスによるエラーを招く可能性がある。よって、エラーを防ぐ意味でモジュール仕様書には使用するタグ定義名が必要とする最低限の引数の数を「最低引数数」として表記することとした。各タグの定義名を覚えていないうちは仕様書で確認出来るようにするためである。

(2) モジュールの基本構造

ホームページを表示するための最低限の HTML 構成は HEAD と呼ばれる頭の部分と、BODY と呼ばれるブラウザで表示される部分、そして BODY が終了し HTML を終了する宣言の 3 つに分けることが出来る。

1. HEAD 部　meta 情報などの部分
2. BODY 部　Web サイトの中身
3. HTML 終了部　HTML が完結したことを示す

この 3 つをまずは最低限の手順で処理できるように作る。その後に BODY に入る各種 HTML タグを表記するものを作っていった。各タグの定義名については HTML のタグと同じ感覚で使用できるよう定義

の呼び出し名を同じにしたが、一部差し替える必要性があった。この事については後述することとする。

(i) HEAD 部（HTML の冒頭部分）

基本的に HTML 文書を作成する場合、同一のホームページ内において冒頭の内容は同じになる。このことを踏まえ、作成するモジュール（html.py）では以下のように書き込まれるよう設定した。（資料編 html.py　416-450 行を参照）

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www/w3.org/TR/html4/loose.dtd">
<html lang="ja">
<head>
<meta HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=Shift_JIS">
<meta HTTP-EQUIV="Content-Style-Type" CONTENT="textcss">
</head>
<body>
```

基本的に日本語で製作した HTML に必要な頭の部分を一度の命令（head という名前で作成）で書き込めるようにした。注目して欲しいのは”-//W3C//DTD HTML 4.01 Transitional//EN”の部分である。この意味を汲み取ると HTML4.01 に準拠し、Web ブラウザにおいて標準モードで動くように命令してある。これにおいて本モジュールを使用し作成した HTML は 4.01 より前のタグを使用しても基本的にブラウザが無視するようになっている。これにより条件であった HTML4.01 準拠を HTML の作成物（.html として作るもの）にも強制している。

（ii）BODY 部（HTML の BODY タグ内の処理）

このモジュールの基本システムとして、all という変数に HTML にするための内容を書き足していき、最後に変数 all を出力することで HTML を作成することにした。（資料編　default.py）ここで問題になるのが改行である。HTML を作成するさい、手動で入れていた改行をどのように処理するかという問題である。

改行はどのようなパターンで行われているか HTML ソースを調べた。結果、属性を付加し長すぎるタグなどを改行することもあるが、多くはブラウザに表示される内容と同じようなデータの並びになるように改行を入れていた。このようなパターンを作成する要素として HTML にはインライン要素(12)とブロックレベル要素(13)がある。タグにはこの要素の特性を反映し、HTML として出力するさいにブロックレベル要素は後ろで改行が入るように処理を施した。（資料編　html.py 262-271 行の定義名 div と 477-498 行の定義名 img を比較）

定義名 `div`

```
262:  def div(s, ID="", CLASS="", STYLE=""):
263:    tag = '<div'
```

・・・・・・途中省略

```
270:    tag += '>\n' + s + '</div>\n'
271:    return tag
```

定義名 `img`

```
477:  def img(src, alt, LONGDESC="", WIDTH="", ・・・
```

・・・・・・途中省略

```
495:    if STYLE !="":
```

```
496:      tag += ’ style="’ + STYLE + ’"’
497:    tag += ’>’
498:    return tag
```

両者の 270 行と 497 行はいずれも終了タグを入力する。だが記号>のうしろをみると 270 行は\n という改行記号がいれてある。この記号を使い、HTML として出力するさいに改行が行われるようにした。

(iii) HTML 終了部

ブロックレベル要素として<body>タグも<html>タグも定義されていない。しかし多くの HTML ソースでは前後は改行してあり、またブロックレベル要素の定義からしてもこの二つはブロックレベル要素として処理するべきである。よってこの二つの終了タグも前後に改行が入るように作成した。(資料編　html.py　307-309 行の定義名 end)

(3) 基本操作

ここまでで HTML の基本構造は作成できるように作った。よって基本操作を検証することにする。

テキストエディタを開き、以下のように入力する。

資料編　default.py

```
1:  #!/usr/bin/env python
2:  # -*- coding:Shift_JIS -*-
3:
4:   from html import *
5:   title("TEST")
6:
```

```
7:  all = head()
8:
9:  all += end()
10:  print all
```

1 行目で Python プログラムであることを示し、2 行目で文字コードが Shift-JIS であることを示し、4 行目で html.py（作成したモジュールを取り扱う）を命令として使用できるように設定している。5 行目からがモジュールを使用し作成する HTML を作成する部分である。5 行目　title() において HTML 文書の title を決めるようになっている。今回は TEST としてある。7 行目　all = head() において変数 all に HEAD 部を代入し、9 行目　all += end() で HTML 終了部を変数 all に加えて終了。この状態で HTML に変換するとブラウザに表示はないが HTML 文書になっていることが確認できる。HTML 文書の核となる本文の部分は 8 行目に当たる位置から変数 all に代入し続けていくことで作成される。

引数という単語についてここで説明をいれておく。5 行目 title() の () の内側に文字列”TEST”が存在する。この場合、定義名 title に対しての第 1 引数は TEST ということになる。続けて「,」で区切って入力していくと右にいくごとに第 2、第 3 引数と呼び名を変えていく。関数の引数処理の説明で書いたが、このモジュールは基本的に第 1 引数に入る値は何を示すか決まっており、第 2 引数以降は属性名を大文字で表記した物を引数名とし、その引数名に値を代入することで属性をタグに追加していく。

例

```
7:  all = head()
8:  all += em("強調したい文字",STYLE="text-align: left")
9:  all += end()
```

default.py に仮にタグを入力した場合の一例を上に表示した。第 1 引数はタグに囲まれた領域を示すので「強調したい文字」はタグに囲まれる。第 2 引数(14)は引数名 STYLE を指定している。これは HTML の属性 style を入力するための引数名を呼び出すものでこれに「text-align: left」を引き渡している。よって 8 行目を HTML として表示すると下記になる。

```
<em style="text-align: left">強調したい文字</em>
```

第一引数の”強調したい文字”がタグに囲まれ、さらに属性 style を指定した状態が作成される。

(i) 入れ子構造

第一引数がタグに囲まれた位置に入ることは説明した。次にタグに囲まれた位置にタグを入れるための「入れ子構造」について動作を検証する。

HTML において一番多く使われるであろう入れ子構造は TABLE タグによる入れ子だと思われる。この構造を再現、検証するとともに製作目的でもあった「HTML の構造が見た目で分かる」ための表記ルールを決めていく。

例）HTML の入れ子構造

```
1:  <TABLE border="1">
2:  <TBODY>
```

```
3:  <TR><TH>自己紹介</TH><TH>じこしょうかい</TH></TR>
4:  <TR><TD>  名前</TD><TD>なまえ</TD></TR>
5:  <TR><TD>  年齢</TD><TD>ねんれい</TD></TR>
6:  <TR><TD>  趣味</TD><TD>しゅみ</TD></TR>
7:  <TR><TD>  住所</TD><TD>じゅうしょ</TD></TR>
8:  </TBODY>
9:  </TABLE>
```

テーブルのタグ構造を示すと<table>の内部に<tr>を配置し、その中に<td>（または<th>）を同一階層で配置している。また<tr>同士も同一階層である。上記の状態を Python モジュールを使って表記する。

例）モジュールを使った入れ子構造

```
1:  all += table(
2:    tbody(
3:      tr(th("自己紹介") + th("じこしょうかい"))
4:      +tr(td("  名前") + td("なまえ"))
5:      +tr(td("  年齢") + td("ねんれい"))
6:      +tr(td("  趣味") + td("しゅみ"))
7:      +tr(td("  住所") + td("じゅうしょ"))
8:    )
9:  ,BORDER="1")
```

変数 all に定義名 table を加える処理だけだが、その定義名 table の第 1 引数には定義名 tbody、さらに tbody の第 1 引数に定義名 tr という形で中にどんどん入れ子構造が進んでいる。定義名 tr の第 1 引数には定義名 th（または定義名 td）が二つ並んで入力されている。この定

義名 th のように同じ階層のものは + 記号を使って連結することで、第一引数の中にいくつもの定義名を入力することが可能になる。

入れ子構造を Python モジュールを使ったソースコードでは、上位にある親のタグと下位に位置する子のタグは改行とインデントを使って表現している。親である定義名 table は子である定義名 tbody を第 1 引数に入れるが、その際に改行しインデントで一段下げた位置から始める。この際に定義名 table の閉じ括弧だけはもう一度改行を行った上でインデントを解除する。こうすることで閉じ括弧の位置関係によってどの定義名の閉じ括弧であるかが分かる。閉じ括弧の存在する頭の位置（例の 9 行目　「,」が閉じ括弧の頭の位置になる）から上をたどり、一番初めに当たる定義名が閉じ括弧の対応する定義名である。

しかし、定義名 tr は親であるが子の定義名 th に対して改行、インデントを行っていない。これがポイントになる。親子関係が存在しても子に当たるタグにブロックレベル要素が無い場合は改行とインデントを行わない。実際には、行っても動作するが逆に見難くなることが多かった。

見やすい入れ子構造をつくるためのルール

1. タグの関係を親子関係で見た場合、子にブロックレベル要素があれば改行、インデントを行う
2. 同じ階層のタグがブロックレベル要素であった場合は改行を行う

上記 2 つは目安であり、そのほかの要因によって改行やインデントはありうるため、絶対条件とはしない。

4　問題点と解決策

動作テストとして実際に存在するホームページを模倣しモジュールの完成度を上げるとともに省力化についても検証をした。模倣を行ったホームページは「大谷大学人文情報学科ホームページ[(15)]」「宮内庁ホームページ[(16)]」「草津市ホームページ[(17)]」の 3 つである。これらのホームページを選んだ理由として HTML 文法チェック[(18)]において 80 点以上を記録し、FLASH [(19)]を使っていない点である。モジュールを使っての FLASH の対応はしていないので見本とすることは避けた。また文法チェックにおいても高得点のものを見本とすることで自ら作成した模倣ホームページが文法ミスが無いことを確認し、また見本側での減点対象が少ないことで模倣する作業がスムーズに進むようにするためでもある。HTML4.01 に準拠した HTML を作成できることが目標のため、文法チェックで点数が低いものは評価がしにくいため除外した。

(1) JavaScript の非対応

問題解決していく上で問題があり仕様を変更することにした。JavaScript への非対応である。正確には HTML 文書内における JavaScript を書く`<script>`タグを使用できなくした。

`<script>`タグは HEAD 内に配置するべきタグだがこれは JavaScript のソースをそのまま HTML 文書に書き込む行為である。多くのホームページで JavaScript を使用する場合ソース内容は 10 行程度から 20 行を超えるものまで多岐にわたっている。そもそもが省力化と分かりやすいレイアウトの実現を目指したモジュールの意義から HTML に直接関係の無いソースを書き連ねることは目的から外れてい

る。別個に作成した JavaScript ファイルにリンクし、対応させるという選択もできたが一方の方法だけを使えるようにするのは使用者に混乱を招くとの判断から、福田先生と相談し JavaScript に対応しないこととした。

(2) Frame タグへの非対応

<frame>タグはブラウザ画面を一画面でありながら幾つかに区切って操作可能にすることが出来るタグである。しかしながら、CSS の登場により画面内をこのタグによって区切らなくとも表示が出来るようになった。また、<frame>タグはブラウザによるバグもあり表示が個人の環境によっても違うというような問題がでており、一般的に使用の推奨はなされなくなった。CSS を使用すれば表示が個人によって代わるなどの問題が少なくすみ、また、長期的にソースコードを利用していく際のメンテナンスも楽になる。以上のことから、<frame>タグに関する項目は非対応とした。

1. <frame>タグ
2. <iframe>タグ
3. <noframe>タグ

Frame に関する非対応タグとは以上の 3 つである。

(3) HTML タグ名と定義名が違うもの

(i) del タグと定義名 delete

実際に存在する HTML を模倣する前段階として、すべての HTML タグが使用できるかを確認するためテストファイルとしてすべてのタグを使った HTML 文書を作成した。このファイルを作成して分かったこ

とが、一部のタグが定義した名前では動作しないという事実だった。

原因として Python に予め用意されている関数 del とタグを使用するための定義名が同一の「del」であり、処理が出来ないというものだった。代替案として単語の意味が通じる範囲で別の定義名（delete）に差し替えることとした。（資料編　html.py　236-249 行の定義名 delete）

（ii）input タグの派生

<input>タグは入力する type 属性によって使うことが出来るそのほかの属性が変化する。しかし初期の作成段階では、すべての属性を入力可能にしていた。これは裏を返すと HTML4.01 では使用不可な組み合わせでも入力可能ということである。属性による人的エラーを減らすにはこの構造は不適格であった。よって入力したい type 属性によって定義名を分け、各 type 属性によって入力可能なその他の属性を制限した。場合分けをしたことにより省力化にもつながった。

仕様書にはタグ名と定義名が一致しないものとして別項目を設けて解説し、使用者の利便を図った。（資料編　モジュール仕様書　定義名が HTML と違うタグ　P30）

（4）HEAD 部に入れる情報の処理

ホームページを作成するさい、独自の meta 情報[(20)]を入力するのが普通である。しかし製作初期の段階では HEAD 部は固定された文章を入力するだけで、mata 情報を追加入力できるようには作っていなかった。解決する単純な方法として HEAD 部の書き込む情報を 2 つに分割し、HEAD 部をパーツごとに呼び出せば meta 情報などを追加可能であることが推察できた。しかし、この方法では 1 つの処理が最低でも 2 つに増えることになる。省力化を目指したものとしてはふさわしくな

かった。そこでこれまでの方法に工夫を加え、HEAD 部の処理は一つの命令だが、追加がある場合は追加した状態で出力する形にすることにした。

手順としては関数の引数処理で説明した方法と同じ、属性の引数名に入力があれば文字列を追加するという流れだが、引数名が追加したい情報のタグ要素という意味に変わるものである。（資料編　html.py 9-49 行と 416-450 行の定義名 head）

例として meta 情報を追加する工程を説明する

```
14:_meta =""
・・・途中省略
17:def meta(s):
18:  global _meta
19:  _meta += '<meta ' + s + '>\n'
・・・途中省略
416:def head(CLASS="", ID="", STYLE=""):
417:  head_str = '''<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transiti
・・・途中省略
423:  if _meta !='':
424:    head_str += _meta
```

14 行でグローバル変数[21]_meta を空文字列として定義する。定義名 head（416 行）が使用されると 423 行でグローバル変数_meta が空文字列でないなら 424 行でグローバル変数_meta の中身を出力することになる。変数_meta の中身を決めるのが 17 行の定義名 meta である。

定義名 meta が使用されると、18 行でグローバル変数_meta を使う宣言を行い、19 行で第一引数を HTML に加工し、グローバル変数_meta に代入する。

meta 情報を追加したい場合、定義名 meta を使いグローバル変数_meta の中身を代入させると、定義名 head ではグローバル変数_meta のデフォルト値が変化した状態になる。この状態で定義名 head を使用すると 423 行で変数_meta がデフォルト値でないために、meta 情報が追加する作業を行なった上で HEAD 部の処理を完了する。

このようなデフォルト値を条件にするプログラムを行うことで必要な場合のみ命令を追加すること処理ができるようになる。

5　結論

(1) ユーザーテスト

2007 年 11 月時点での製作途中ではあるがモジュールの動作テストを人文情報学科福田ゼミ 3 回生の 5 人に行ってもらった。この 3 回生たちはいずれも Python を授業において使用し、また HTML をある程度理解している生徒だったのでテストケースとして最適であった。モジュールを自分以外の人が使用することでモジュールの問題点、改善点など発見することが目的である。

テストはモジュールの簡単な動作方法を説明したものをプリントで渡し、あとは各自に HTML 文書を作成してもらうという形で行った。アンケートには使用した時に感じた難易度、使用方法を習得し引き続き使用したいかという意識、要望などを書いてもらった。

アンケートをとった結果、使用した時に感じた難易度としての集計は

・　まぁまぁ簡単　　2 人
・　普通　　2 人
・　少し難しい　　1 人

使用方法を習得し引き続き使用したいかについては

・　慣れれば使用したい　　5 人

使用感としては普通と呼べるが、HTML を作成する際にこのモジュールを使用したいと肯定的な意見を全員からもらった。

頂いた感想として

・　閉じるタグを省略できるので打ち間違いが減って良い
・　どこにタグが係っているのか分かりやすい
・　命令タグが HTML と同じなので分かりやすい

これらの意見・感想から総合するとモジュールの作成目的は使用者に肯定的に受け入れられたと考えられる。ただし、要望として以下のように

・　説明書のようなものが欲しい
・　タグの使用例がもっと欲しい
・　タグ命令のコマンドが分からない

モジュールを使うには説明不足だという意見を頂いた。この意見を受けて仕様書を改良し、説明をより充実したものとした。

(2) 自己評価

ここまでの製作を終えて自己評価を行うと条件であった上記 2 つ（閉じタグを入力する必要のない HTML の作成、HTML のソースコード構造が見た目で分かる）はユーザーテストの意見からもクリアできたと考える。

そして、最大の評価点である省力化については再現した 3 つのホームページすべてにおいて、文字数が元のデータと比べ減っていることを確認できた。特に単純なタグを多用した人文情報学科のホームページについては 2 割以上の減少を確認できた。しかし、長大なデータでかつ属性を多く指定するものについては著しく差が縮まってしまった。これは閉じタグは省力化できているが属性の入力においては効果が少ないものだからだと推測できる。また文字数が増えたもう一つの要因としてホームページのレイアウトを同じにするために使用できないタグ（例： タグは HTML4.01 準拠ではない）を定義名 div と style 属性によって調整したことにより、著しく文字数が増えたことも影響がある。しかしこの問題は 0 からホームページを構築するさいには問題になりえない。よって影響を加味した上で、モジュールを使った省力化は一応の成功を収めたと考える。

今後の課題としては JavaScript への対応が必要だと考える。現在多くのホームページが JavaScript を利用している。今後ますます利用拡大が予想されるため、JavaScript への対応は考えるべき案件である。次に自動で改行する機能が欲しいと思った。HTML に出力すると Python データは改行しているが HTML ではコマンドでの改行になるため横長に文字列が伸びてしまうことがあった。気がついた際に Python ソースのほうに改行コマンドをいれて凌いだが、この作業をなんとか自動にならないかと考えている。また、省力化についてもさらなる改良を加えて欲しいと思う。

最後にここまで製作して思ったことは、人の役にたつものをつくるということの難しさである。プログラミングも大変ではあったが、人に関わる部分は特に大変だと感じた。つくるという作業は作成するのも人、

作ったものを使うのも人である。作成途中にプログラムで他人に助言を仰ぐ時に、どのように自分の考えていることを伝えれば相手に理解してもらえるのか。利用者のことを考え、どうすれば省力化につながるか、仕様書を分かりやすく作成するにはどうすればよいのかなど、人への配慮は想像以上に難しい作業であった。自分ひとりが使いやすいものを目指すのではなく、知らない人が一人でも利用してもらえるように作る。当たり前だがそれだけに難しいということを体感した。

注

（1）コンパイルする必要の無いスクリプト言語

（2）HTML=HyperTextMarkupLanguage

（3）文章をタグと呼ばれる特別な文字列で囲むことにより文章構造、見栄えなどを表す

（4）テキストファイルを作成、編集、保存するためのソフトウェア

（5）HTML のタグは開始タグと終了タグで文章を囲むことで意味をなす

（6）プログラムの元となるテキストデータ

（7）W3C（Web の標準化団体）の定める HTML の表記ルール

（8）システムを構成する要素となるもの　小さなプログラム

（9）タグに付加する情報

（10）PyJUG
(http://www.python.jp/Zope/download/pythoncor)

（11）ユーザに対する情報の表示を文字によって行ない、すべての操作をキーボードを用いて行なうユーザインターフェースのこと

（12）文章の一部としてブロックレベル要素の中で使用される要素

（13）ひとつのまとまった単位としてあらわされる要素

（14）引数名を指定しているので正確には第二引数とは呼ばないが位置を示す言葉として使用した

（15）http://www2.otani.ac.jp/hi/

（16）http://www.kunaicho.go.jp/

（17）http://www.city.kusatsu.shiga.jp/

（18）Another HTML-lint gateway
(http://htmllint.itc.keio.ac.jp/htmllint/htmllint.html)

（19）アニメーションと音を組み合わせて動的な Web コンテンツを作るソフト

（20）HTML 文書に関する情報

（21）プログラムの中のもっとも外側で宣言された変数

文献表

大藤幹

2007 『詳解　HTML & XHTML & CSS 辞典　第三版』
秀和システム

Mark Lutz　&　David Ascher

2004 『初めての Python　第 2 版』
オーム社

柴田淳

2006 『みんなの Python』
ソフトバンククリエイティブ株式会社