経営論集　第 20 巻第 1 号　2010 年　103 ～ 125 頁

HTML+CSS の速習カリキュラムの提案

小　松　香　爾

はじめに

本論文では、HTML と CSS のカリキュラムを提案する。カリキュラムの対象は標準的な大学生である。また、半期科目であることを前提とする。

近年、数多くの CMS が登場した。ブログもその一つであり、小学生でも Web ページをインターネットに公開できるようになった。しかし、CMS にはいくつかの問題点がある。最大の問題点は、サーバに CMS がインストールされているか、ユーザがインストールしなければならないということである。CMS を導入するとサーバの負荷があがり、レスポンスが遅くなる。したがって、CMS のインストールを禁止しているサーバも存在する。

DreamWeaver に代表される Web オーサリングソフトは、Web サーバにはインストールされないという点で CMS とは異なる。Web オーサリングソフトによって生成されたコードは、徐々に Web 標準に従ったものになり、CSS への対応も進んだ。しかし、Web オーサリングソフトは、プログラミングにおける Eclipse に相当する存在である。つまり、サイト制作の効率を高めたり、チームによる分業制作を実現したりするためのツールである。HTML と CSS の知識がない学生が、使いこなせるアプリケーションとしては設計されていない。

Web サイトはエディタさえあれば制作できる。Web のコンテンツを記述するための言語が HTML であり、ページのデザインを変更する為の言語が CSS である。Web 関係の仕事をするためには、これらの知識を最初に身につける必要がある。本論文では、大学における HTML および CSS を、できるだけ迅速に習得するためのカリキュラムを提案する。

Web に限らず、学習においては「基礎が重要」と言われる。基礎が重要なのは疑いない。しかし、「何を基礎とするか」を提案しなければ、この言葉には実質的な意味がない。これまでの教育経験から述べると、HTML と CSS の要素、属性、プロパティを網羅的に演習させた場合、実用レベルの Web ページの制作ができるようになる学生は、1 割にも満たない。本論文では、「基礎は実務の中で使えてこそ意味がある」というポリシーの基づき、実際の Web 制作の現場で必要な知識に焦点が当てられる。そして、実用性、簡潔さおよび学習速度を重視したカリキュラムを提案する。

1　HTML の速習カリキュラム

Web サイトは、コンテンツありきである。伝達したいコンテンツがあって、コンピュータネットワークで共有する。そのための言語として HTML が考案された。したがって、Web サイト

制作で最も重要なことは、プレーンなテキストを適切に「マークアップ」することである。最初に教えるべき事は、マークアップすることの意義である。文書をネットで公開する際に、プレーンなテキストは稀であり、ほとんどが Web ページである理由を考えさせる。テキストが適切にマークアップされていれば、見る側の環境に依存せず、ブラウザで見やすくレンダリングされる。HTML のタグを使ってマークアップされたテキストは、単なるテキストではなく HTML の要素の 1 部になる。空要素はあくまで例外的な要素であり、「元となるテキストが存在しないため、空要素としてマークアップすること」に注意させる。

上記の流れで教えれば「HTML ファイルをテキスト形式で保存する」という間違いも起こりにくい。しかし、保存の際には、ファイル形式以外にも注意すべきポイントがある。漢字コードである。Web サービスが熟成し、Web アプリケーションで扱いやすい UTF-8 が主流になりつつある[(1)]。また、Unicode には、一つのファイルに、複数の非英数字を含められるというメリットもある。日本語版 Windows では、漢字コードとして、ほとんどのアプリケーションでシフト JIS が使われてきた。したがって、デフォルトの状態では、シフト JIS 形式で保存されてしまう。エディタで書いた HTML ファイルを保存する際には、文字コードを UTF-8 に変更させる必要がある。また、Unicode を使用すれば、HTML の xml 宣言を行わなくてもよい。これは大きな利点である。なぜなら、xml 宣言を行うと、IE6 以前のブラウザのレンダリングモードが過去互換モードになってしまうからである。過去互換モードは、古いブラウザとの整合性を取るために、マイクロソフトが独自に実装した機能である。ブロック要素をセンタリングする際に、本来、インライン要素のセンタリングである「text-align: center」でなければセンタリングされないことや、ブロックボックスの幅の解釈の違いなど、学習者を惑わせる様々な問題がある。ただし、IE を Web 制作の教育で使うこと自体に問題があるともいえる［1］。しかし XP がいまだに多く残っている現状では、全く無視するわけにもいかない。したがって、xml 宣言なしでもエラーにならないという点においても UTF-8 形式を選択させるべきである。

現時点では HTML5 はまだ正式勧告されてない。したがって、演習用の HTML としては、XHTML1.0 か HTML4.01 を選ぶことになる。xml 宣言の問題が解決できるならば、XHTML を選択したほうが無難である。XHTML は SGML で定義された HTML を XML で再定義したものである。厳密性の面で SGML よりも XML の方が高く、コンピュータで自動処理しやすい[(2)]。XHTML 文書は XML 文書としても HTML 文書としても扱える。現在では、XML 文書として処理される機会はすくない。しかし、将来、XML 文書として自動処理される技術が出現する可能性もある。

1-1　HTML のマークアップ

HTML のマークアップ方法は単純である。HTML の要素には、通常の要素と空要素がある。通常の要素は「< 要素名 > コンテンツ </ 要素名 >」と、開始タグと終了タグを使ってマークアップすることから教える。要素には、属性が指定される場合があり、開始タグに「< 要素名 属性名 =" 値 ">」と記述される。空要素は、特別に 1 つのタグでマークアップされ「< 要素名 />」

となる。これは、開始タグでも、終了タグでもない。開始タグと終了タグの組がまとまって省略されたものである。「< 要素名 ></ 要素名 >」と書いても間違いではない[(3)]。次に、通常の要素以外に、コメント、文字実体参照、文書型宣言が存在することを教えるべきである。

HTML のコメントは「<!-- コメント -->」と書き、CSS のコメントは「/* コメント */」であることも同時に教えてしまったほうがよい。コメントの形式が紛らわしいからである。特に HTML では、一般のプログラミング言語のコメントとは異なる記述方法が採用されている。それにも関わらず、HTML のコメントアウトの方法を教える必要があるのは理由がある。div 要素は開始タグと終了タグの間が長くなりやすく、div 要素の前や後に、コメントを入れることによって、div 要素の終了タグを忘れるなどのミスを減らせるからである。

文字実体参照は、非常に数が多い。しかし、特殊な記号は全角で書けばよいので、それほど使う機会はない。時間を割いて教える必要はないといえる。唯一、半角空白は、DreamWeaver で頻繁に出現する。したがって「 」だけ、教えればよい。

XHTML の文書型宣言は何種類かある。XHTML1.1 では、要素が整理されモジュール化が実現された。ケータイ端末のブラウザのサイトは作りやすくなったといえる。しかし、新たに追加された要素はルビ要素のみであり、パソコン向けの Web サイト制作においては利点がない。また、MIME タイプとして「application/xhtml+xml」が推奨されているが、IE が、ダウンロードファイルとして解釈してしまうという問題もある。したがって、現時点では、XHTML1.0 を選択することが妥当である。XHTML1.0 の STRICT DTD を用いるならば、「<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">」と書かせる。HTML のみで、テーブルレイアウトを行う場合は、STRICT 型では規則違反になる属性を使わざるを得ない。したがって、「<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">」として、TRANSITIONAL 型を書かせる。

1-2　含めるべき最小限の要素

最小限教える要素は、html、head、title、body、p、h1 ～ h6、a、ul、li、dl、dt、dd、img、div、span、table、tr、td の各要素である。

html、head、body の 3 要素がなければ html 文書にさえならない。これらは構造要素とも呼ばれる。また、title 要素も構造要素に準ずるものであり、決して省略させてはならない。title 要素がないと、ユーザビリティが著しく損なわれるからである。title 要素のコンテンツは、ブックマーク時のタイトルになる。また、近年のタブブラウザでは、タブに表示されるタイトルにもなる。インターネットで公開した場合には、検索エンジンのキーワードとしても使われるので、SEO 対策にもなる。

p 要素が必要な理由は文法的な制約にある。XHTML では、body 要素の子要素は、ブロックレベル要素でなくてはならない。したがって、div 要素か p 要素がなければ、段落をマークアップすることすらできない。div 要素は、要素をグループ化するための要素であるので、段落

をマークアップするために使用するのは好ましくない。

h1 ～ h6 の要素は見出しを作る要素である。段落の前には見出しを置いた方がユーザビリティがよい。また、検索エンジンのロボットは見出しのキーワードを重視する。したがって、キーワードを見出しに含めることは、SEO 対策としても有効である。

a 要素は、ハイパーリンクを生成する要素である。ハイパーリンクが埋め込まれている文書がハイパーテキストである。a 要素は、HTML（Hyper Text Markup Language）の語源ともなっている要素である。また、複数のページからなるサイトを制作する際に必要な要素である。実際、ポータルサイトの HTML 文書はリンクのかたまりである。サイト内にリンクを設定する場合は、パスの概念が必要になる。しかし、全ての HTML ファイルを一つのフォルダに入れれば、パスはファイル名を記述するだけである。

ul 要素と li 要素は、リストをマークアップするため要素である。グローバルナビゲーションやサムネイルの並びなどは、リスト項目が並んだものとしてマークアップし、CSS でレイアウトするのが妥当である。

dl、dt、dd の各要素は、定義リストをマークアップするための要素である。サムネイルとその説明文など、二つの内容を列挙する場合に使用するのが妥当である。

img 要素は、画像を入れる際に使う要素である。近年の回線の高速化によって、画像が頻繁に使われるようになってきた。画像をページに貼る場合も、リンクと同様に、パスの問題が生じる。しかし、HTML ファイルと同じフォルダに画像を置けば画像のファイル名を書けばパスになる。また、img 要素はコンテンツとして画像をマークアップしたい場合だけに使用するべきものである。意味のない装飾的な画像は、CSS の background プロパティなどを使って背景としてデザインするべきであることを教える。

div 要素はブロックレベル要素をグループ化するための要素である。HTML 単独で済ませる場合は必要とはいえない。しかし、CSS でデザインする場合は必要になる。なぜなら、p 要素の子要素として、ブロックレベル要素を入れることはできないからである。したがって、p 要素を div 要素の代替として使用することはできない[(4)]。

span 要素はテキストやインライン要素をグループ化するための要素である。div 要素と同じく、CSS でデザインする場合に必要になる。

table 要素、tr 要素、td 要素は、表をマークアップするための要素である。div 要素や p 要素に CSS をうまく適用すれば表を組むことはできる。本論文で、table 関係の要素の導入を薦めるのは、テーブルレイアウトに使えるからである。テーブルレイアウトは、本来の table 要素の意味から外れるため、好ましい方法とは言えない。

1-3　テーブルレイアウトの是非

テーブルレイアウトの是非については、未だに議論が尽きない。table 関係の要素は、本来、表組みを行うためのものである。レイアウトするための要素ではない。レイアウトに table 関係の要素を用いることは、適切なマークアップにはなり得ない。アクセシビリティでも問題があ

る。音声ブラウザはファイルの上から下へとコンテンツを読みあげていく。したがって、アクセシビリティに考慮するならば、重要なコンテンツを上に書くべきである。ところがテーブルレイアウトでは、それができない場合がある。また、欧米では、テーブルレイアウトのサイトが、ほとんど CSS レイアウトに移行したのも事実である。しかし、日本の企業サイトは、未だにテーブルレイアウトのサイトが半数近くを占めている。日本の Web 制作業界の事情は特殊である。日本では伝統的に出版業が盛んであった。したがって、Web 制作が DTP の延長と見なされてきた。Web 制作者の中には、DTP 業界にいた者も多い。DTP 業界の人材はデザイナーであってプログラマではない。ところが、CSS には、同じレンダリング結果を生むのに多数の書き方があるなど、プログラム的な部分が多い。また、CSS は、1996 年の勧告から 5 年間は、ブラウザの実装状況が良くなかった。特に 2000 年以前は絶望的な状況であった。現在でも、以前ほどではないが、ブラウザごとのデザインの違いは存在する。近年、CMS の登場もあって、デザイナーも CSS を学ぶべきという風潮が強まってきた。テーブルレイアウトを採用してきた日本の Yahoo! も、2008 年の 1 月についに CSS レイアウトに移行した。したがって、もはやテーブルレイアウトを教える必要性はないという見解も妥当である。本論文で、あえてテーブルレイアウトの導入を薦める根拠は、教育的効果にある。1 つ目は、CSS レイアウトの導入として使えるということである。「XHTML1.0 Transitional」を用いれば、HTML 単独レイアウトすることができる。2 つめは、CSS レイアウトの優位性が明確になるということである。テーブルレイアウトは CSS レイアウトと比べると HTML のソースコードが煩雑になる。通信量の増加は無視できるとしても、コンテンツの更新が困難になることは問題である。編集対象であるコンテンツが、どのセルに入っているのかが分かり難い。しかも、レイアウトを変更する場合は、煩雑なソースコードを編集する必要がある。CSS の最大の利点は、デザインとコンテンツの分離である。CSS レイアウトならば、HTML のソースは簡潔である。さらに、HTML のソースを編集することなしに、レイアウトの変更が可能である。

1-4　HTML の演習

HTML は、コンテンツありきである。したがって、マークアップするためのテキストを用意させておく。Wikipedia のテキストが、ある程度の長さがあるので、演習用に適する。ただし、Wikipedia は著作権フリーではなく、GFDL（GNU Free Documentation License）である。もし、公開する際には、GFDL を守らなければならないことに注意させる。

マークアップに際しては、文書型宣言や構造要素などから入力させる必要はない。しかし、既存の Web ページのソースを表示させて、それをコピーする方法をとらせるよくない。meta 要素や body 要素の内容を大幅にカットする必要があるからである。必要最小限の HTML 要素を半自動生成するサイトを用いるのがよい［2］。

HTML にエラーがあると、CSS を書いても、仕様通りのデザインが適用されない。したがって、HTML のエラーチェックは不可欠である。Web サービスとして、いくつかのページが公開されているので、それらを利用させるのがよい。以下に HTML の演習の手順を示す。

①「XHTML テンプレート」で検索、以下のページを開く

http://heaven7.sakura.ne.jp/reference/xhtml_ref/xhtmltmp.html

②「XML 宣言」のチェックをはずす

③「DOCTYPE」は「XHTML 1.0 Strict」をチェック

④「文字コード」は「Unicode UTF-8」をチェック

⑤「スタイルシート（CSS）を使う」をチェック

⑥「テンプレートの作成」ボタンを押す

⑦作成された HTML のソースをエディタにコピー＆ペースト

⑧ title 要素のコンテンツを書き換える

⑨ p 要素のコンテンツとして、用意したテキストをコピー＆ペースト

⑩ HTML でマークアップ

⑪ファイルの種類を「HTML 文書」、文字コードを「UTF-8」にして保存

⑫「HTML 検証」で検索して、以下のページを開く

http://openlab.ring.gr.jp/k16/htmllint/htmllint.html

⑬「DATA」をチェックして、テキストエリアに HTML ソースをコピー＆ペースト

⑭「チェック」ボタンを押してエラーチェック

上記の手順を一通り演習させるのに、1 時間もあれば十分である。ただし、⑩のマークアップでは、h1 ～ h3、p、div 要素しか使わないものとする。図 1 に示される①の検証サイトは、ユーザが各種初期設定項目を選択する方式をとっている。このサイトを使用して、推奨する手順に沿って生成した HTML ファイルの雛形が図 2 である。ユーザは「●文書のタイトル●」を書き換え、「●文書の内容●」にテキストを流し込むことになる。図 3 に示される⑫の検証サイトは「Another HTML-lint」と呼ばれる。稼働期間は 13 年以上でありアップデートやバグフィックスも細かく行われてきた。Web 制作者の間でも非常に評価が高い。その理由は、エラーチェックの細かさにある。W3C の検証ページ「W3C HTML Validation Service」では、DTD で定められた規則に従っているかどうかがチェックされるだけである。「Another HTML-lint」では、仕様書には書かれているが DTD には現れてこない条件までチェックされる。その結果、詳細かつ親切なエラー表示がでることになる。図 2 の HTML をエラーチェックにかけた結果が図 4 である。結果が 100 点になっていないのは、文書型を XHTML にしたにもかかわらず、xml 宣言を省略したからである。

文書宣言	
☐ XML宣言	XMLの宣言（[文字コード]に影響されます）
☐ DOCTYPE	ドキュメントのバージョン
◉ XHTML 1.0 Strict	厳密
○ XHTML 1.0 Transitional	寛容
○ XHTML 1.0 Frameset	フレームセット
○ XHTML 1.1	最新のXHTML
○ XHTM L Basic	小型情報機器向け
コンテンツ宣言	
☑ 文字コード	文字コードの宣言（[XMLの宣言]に影響を与えます）
○ シフト JIS	Shift_JIS
○ JIS	ISO-2022-JP
○ EUC	EUC-JP
◉ Unicode	UTF-8
☐ JavaScriptを使う	文書でJavaScriptを使う
☑ スタイルシート(CSS)を使う	文書でスタイルシートを使う
文書の説明	
☐ 著者名	文書の著者名
☐ 著者連絡先	著者のEメールアドレス
☐ 内容説明	検索ロボットなどに渡す文書の説明
☐ キーワード	検索ロボットなどに渡す文書のキーワード
ナビゲーション	
☐ 目次	
☐ 索引	
☐ 一つ上の文書	ナビゲーション情報 一部のブラウザではこれらの情報からナビゲーションボタンを表示するものがあります
☐ 最初の文書	
☐ 前の文書	
☐ 次の文書	

図1　テンプレート生成サイト

http://heaven7.sakura.ne.jp/reference/xhtml_ref/xhtmltmp.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
          <meta http-equiv="Content-Style-Type" content="text/css" />
          <title>●文書のタイトル●</title>
</head>
<body>
          <p>●文書の内容●</p>
</body>
</html>
```

図 2　生成された HTML のソース

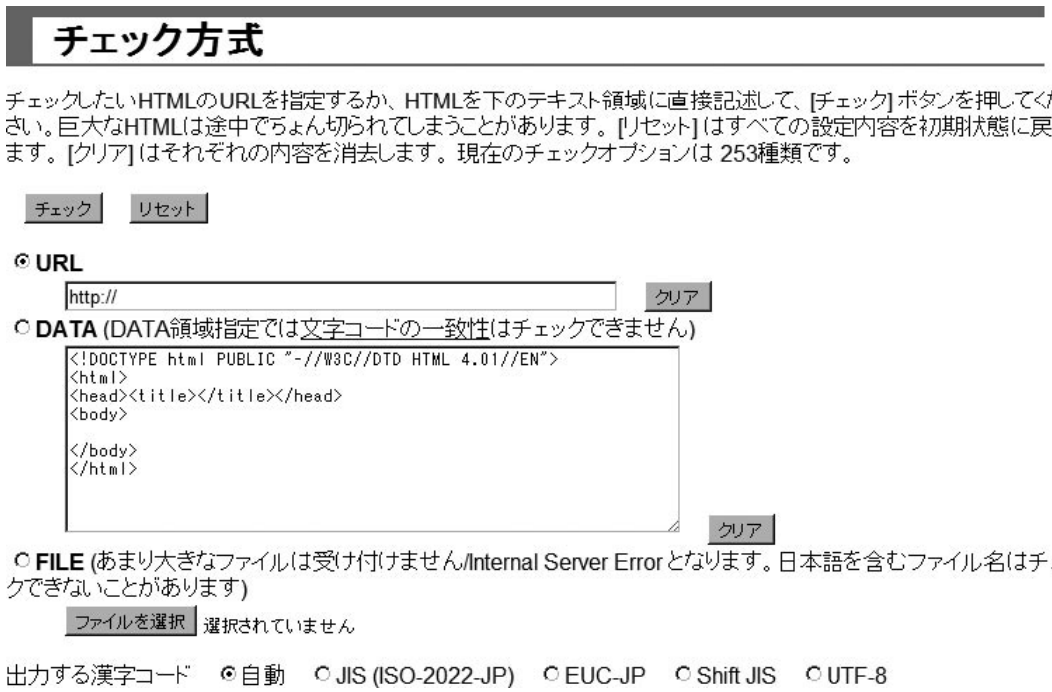

図 3　HTML の検証サイト

http://openlab.ring.gr.jp/k16/htmllint/htmllint.html

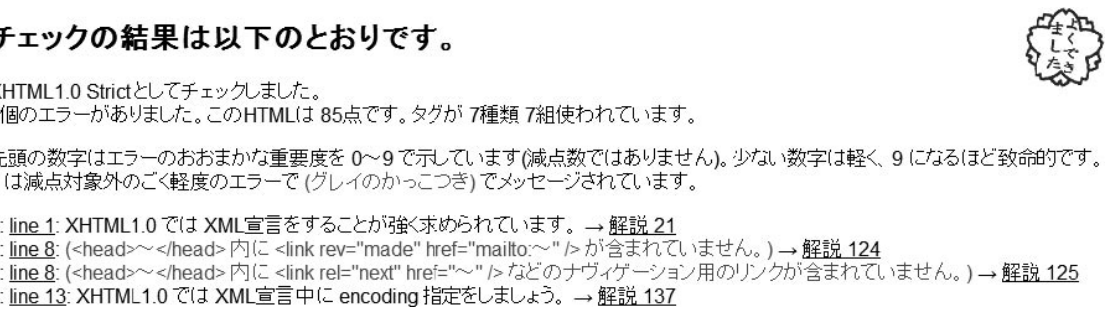

図 4　HTML 検証サイトによるエラーチェックの結果

2　CSS の速習カリキュラム

CSS はエラーチェックを厳密にさせる必要がある。HTML は、エラーがあっても表示されないということはない。しかし、CSS では「:」「；」「{」「} 」などの記号が一つ抜けただけで、それ以降のデザインが一切適用されなくなる。目に見えないものの修正は困難である。したがって、HTML と同様に、CSS の演習でもエラーチェックは必要である。

CSS で、最初に教えるべきことは、ブラウザスタイルシートの存在である。ブラウザに HTML 要素がレンダリングされるときに、ブラウザ固有のスタイルシートが適用される[5]。した

がって、HTML ファイルを作った時点で、ブラウザのデザインが適用されることになる。デザインにこだわりがなければ、CSS を使う必要はない。正確に言えば、CSS は「デザインするための言語」ではなく、「デザインを変更するための言語」である。したがって、「HTML でデザインできるのに、何故 CSS が存在するか」ということついて教えるべきである。

CSS の存在意義については「構造とデザインの分離」および「デザインを分離したことによるメンテナンスの容易さ」ということがよく言われている。もちろん、その通りではある。しかし、それはある程度の規模のページを作成しないと実感させることはできない。本格的なデザインの分離は、HTML の link 要素や、CSS の @import を使った外部 CSS ファイルの読み込みが必要になる。それらは、どちらかといえば、応用的な演習を通じて実感させるべき性質の事項であり、演習の初期段階で教えるのは無理がある。それよりは「HTML 単独ではできないデザインができるようになる」ということを強調したほうがよい。文字やテーブルの装飾は、HTML でも可能である。しかし、CSS を使えば、ほぼ全ての要素に対し装飾が可能になる。HTML の任意の要素を「border」プロパティで囲んだり、「background」プロパティで背景色を付けたりさせることを、最初の演習として設定するのが適当である。

次に、CSS の記述法、セレクタ、プロパティ、プロパティの値を教えていくことになる。プロパティとプロパティ値の組み合わせは、多数存在する。しかし、頻繁に使用するものやショートハンドプロパティに絞れば、三分の一程度である。その中でも重要なのは、ボックスモデルに直接的に関係するプロパティである。全ての HTML 要素は、ボックスモデルに基づいてレンダリングされるからである[(6)]。

2-1　CSS の記述

CSS の書式は、非常に単純であり、階層的に説明できる。階層のトップは「セレクタ 宣言ブロック」である。セレクタは、文字通り「HTML 要素を選択するもの」と説明する。多数のセレクタを使用する場合は、セレクタをグループ化し「セレクタ , セレクタ , ･･･ , セレクタ 宣言ブロック」と書くことができる。宣言ブロックは、いくつかの宣言がまとまったものであり「{ 宣言 宣言･･･宣言 }」と書かれる[(7)]。宣言は「プロパティ : 値 ;」と定義されている。宣言ブロックはデザインの内容である。プロパティは、HTML 要素がレンダリングされる際の、なんらかの性質を表す。プロパティの値は、初期値が定まっていて、デザインを変更する場合に値を指定することになると説明する。ただし、プロパティごとに書ける値が決まっていることに注意させる。

CSS を読みやすくするために、改行や半角空白を入れることが多い。記述法には様々な流儀がある。宣言が多くなる場合は、宣言ブロックの内部で改行する記述法が多用される。そのとき図 5 のようにプロパティの前に半角空白を入れさせたほうがよい。これは、セレクタとプロパティの区別をつけやすくするためである。

```
セレクタ, ･･･, セレクタ {
  プロパティ: 値;
  プロパティ: 値;
          ･
        ･
  プロパティ: 値;
}
```

図 5　CSS の常套的な記述法

2-2　CSS のセレクタ

デザインを適用する HTML 要素を選択するものがセレクタである。CSS2.1 のセレクタは 12 種類ある。しかし、必要なセレクタは、タイプセレクタ、ユニバーサルセレクタ、クラスセレクタ、ID セレクタ、子孫セレクタ、疑似クラスセレクタの 6 つである。他のセレクタは、セレクタの記述を簡潔にするものや実務では、ほとんど使われないものである。

タイプセレクタは、HTML の要素名そのものである。記述された要素名と同じ要素名を持つ各 HTML 要素にマッチする。もっとも基本的かつ頻出するセレクタであり、最初に教えるべきである。

ユニバーサルセレクタは「*」と書かれる。これは全称セレクタとも呼ばれ、文字通り全ての HTML 要素にマッチする。全てのタイプセレクタを並べたものに等しい。ブラウザスタイルシートをリセットする場合に多用される[8]。

クラスセレクタと ID セレクタを使用する場合は、HTML の属性で、クラスと ID を指定する必要があることに注意させる。クラスセレクタは「. クラス名」と書かれる。同じクラス名が指定されている全ての HTML 要素にマッチする。ID セレクタは「#ID 名」と書かれる。同じ ID 名が指定されている HTML 要素にマッチする。クラスセレクタと ID セレクタは似ているが、使用のされ方は異なる。クラスセレクタは、要素名に関係なく選択したい場合に使用され、ID セレクタは、ある一つの要素だけを選択したい場合に使用されると説明する。

子孫セレクタは、セレクタを組み合わせてつくるセレクタであり、「セレクタ セレクタ･･･セレクタ」と書かれる。HTML 要素のなかに含まれる子孫要素にマッチする。組み合わせるセレクタの種類は問われないこと、子孫セレクタを使えば、クラスの数を減らすことができることに注意させる。

疑似クラスセレクタは、「: 疑似クラス名」と書かれる。疑似クラスは「link」「visited」「hover」「active」「focus」の 5 種類である。しかし、実際には a 要素以外に疑似クラスセレクタを使うことは稀である。a 要素の疑似クラスセレクタは「a:link」、「a:visited」、「a:hover」、「a:active」の 4 つである。また、正常に機能させるためには、上記の順序で指定する必要があることも教える

必要がある。

2-3　CSS のプロパティと値

CSS のプロパティを一通り演習させるのは非常に時間がかかる。値との組み合わせが CSS の仕様で決められているため、プロパティの値も同時に学習させる必要があるからである。しかし、ボックスモデルに直接関係していて、しかもよく使用されるプロパティは限られる。「margin」「border」「border-left」「border-right」「border-top」「border-bottom」「padding」「width」「height」「position」「left」「right」「top」「bottom」「overflow」「float」「clear」「display」である。他に、頻繁に使用されるプロパティは「background」「color」「line-height」「list-style」「text-align」「vertical-align」「font-size」「font-weight」「font-family」である[9]。CSS の速習を目標とする場合は、上記のプロパティに絞って演習させる。各プロパティの値が CSS で明示的に指定されることにより、ブラウザスタイルシートの初期値が書き換えられ、デザインが変更されることを説明しておく。

2-4　CSS の値の単位

CSS のプロパティの値が数値の場合には、注意させる必要がある。基本的に、CSS における数値は、単位をつけなければならないからである[10]。単位が必要でない場合は限られる。値が「0」の場合と「line-height」プロパティで倍率を指定する場合である。教えるべき数値の単位は、長さを表す「px」「em」と割合を表す「%」である。

「1px」はグラフィックスのデバイスドライバで定められる 1 ピクセルの長さであり、レンダリングの最小単位である。「border」プロパティの罫線の幅や、画像の「width」プロパティの指定などで頻繁に使用される。

「1em」は CSS が適用されている html 要素内部の文字「m」の横幅である。文字を基準とする相対的な長さであり、実務では頻繁に使用される。日本人には馴染みがないが、使い勝手がよいため、強調して教えるべき単位である。

2-5　ボックスモデル

body 要素の内部の全ての HTML 要素は、ボックスモデルでレンダリングされることを教える。「width」、「height」プロパティで、内容領域の幅と高さ、「padding」プロパティで内側余白、「border」プロパティで外枠、「margin」プロパティで余白を指定することを最初に演習として取り上げる。図 6 に p 要素に対する典型的な CSS 指定を示す。

全てのボックスには包含ブロックが存在することも重要である。なぜなら「position」プロパティや「float」プロパティの習得には、包含ブロックの知識が必要だからである。「position」プロパティを指定しなかった場合は、「static」という初期値をとる。「position」プロパティに初期値か「relative」が指定された場合、ある要素の包含ブロックは、親要素のコンテンツ領域が作るボックスになる。そして、このとき、当該要素のボックスは包含ブロックの中に上から下へ配置されていく。このようなボックスの配置を「通常フロー」と呼び、デフォルトのレイアウトであることを説明する。通常フローにおける配置では、隣接したボックスの上下のマージン

```
p {
  width:10em;
  height:5em;
  padding:0.3em;
  border:1px solid #000000;
  margin:1em;
}
```

図 6　幅、高さ、内側余白、外枠、余白の指定

は相殺される。図 7 は、二つの p 要素に対して、図 6 の CSS を適用したものである。上のボックスの下マージンが 1em、下のボックスの上マージンが 1em であり、相殺がなければボックス間の余白は 2em となるはずである。しかし、マージンの相殺が発生し、ボックス間の余白は 1em となる。マージンが隣接する場合には、数値が大きいほうのマージンが適用される。数値が小さいほうマージンは相殺によって無視されることに注意させる。

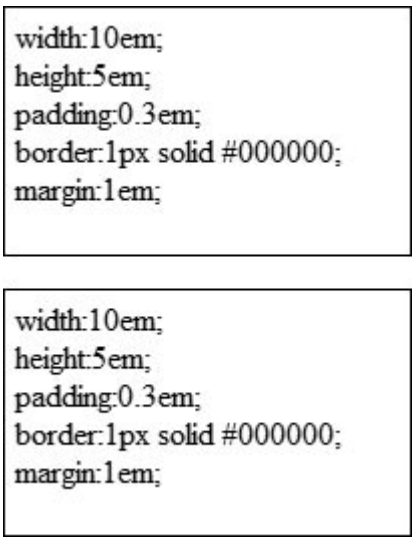

図 7　上下のマージンの相殺

ボックスの配置は、通常フローの他に、「浮動化」と「絶対位置決め」が存在する。浮動化と絶対位置決めは、仕様が複雑であり、初学者が躓きやすいポイントである。「position」プロパティと「float」プロパティの値によって、ボックスの配置が変化するからである。

「position」プロパティに「fixed」が指定された場合、ブラウザの表示領域が包含ブロックになる。ボックスの配置位置は、包含ブロックを基準にして「top」「left」プロパティなどで指定

される。このとき、ボックスはスクロールの影響を受けず、完全に固定されてレンダリングされる。

「position」プロパティに「absolute」が指定された場合、「position」プロパティの値が「static」以外に指定されている要素[(11)]のうち、最も近い祖先要素のパディング領域のボックス[(12)]が包含ブロックになる。そのような祖先要素がない場合は、ルート要素によって作られる初期包含ブロックが包含ブロックになる。このとき、包含ボックスを基準として、「top」「left」プロパティなどの値でボックスの位置が決められる。

「position」プロパティに「fixed」か「absolute」が指定する配置は「絶対位置決め」と呼ばれ、通常フローから外れることになる。絶対位置決めを使えば、ボックスの自由な配置が実現できる[(13)]。

「float」プロパティが「none」以外に指定された場合は、ボックスは浮動化して配置される。このとき、やはり通常フローから外れることになる。

絶対位置決めと浮動化は同時には行うことはできない。どちらも指定された場合は、絶対位置決めが優先され、「float」プロパティの値は無視される。また、どちらの場合も上下マージンの相殺が起きない。通常フローから外れた要素の後に続く要素のボックスは、「width」プロパティの値が指定されて以内場合は、通常フローから外れた要素がなかったかのように、上につめられて配置される。このとき、浮動化して配置されたボックスがある場合、上につめられて配置されたボックスの内部のインラインボックスは、左側か右側に回り込んで配置される。

2-6　浮動化されたボックス

浮動化されたボックスは通常フローから外れて、左右に浮動化されて配置される。その際に、初学者を悩ませる副作用が生じる場合がある。その対処法を教えるべきである。

図 8 に、CSS で浮動化された要素を含む HTML のソースを示す。浮動化された要素の後続のブロックレベル要素は、「width」プロパティの値が指定されていない場合は、図 9 に示されるように、浮動化された要素がなかったものとして上方につめられて配置される。この時、インライン要素が生成するインラインボックスは、浮動化されて配置されたボックスの右側に回り込んで配置される。ただし、図 10 に示されるように、ブロックレベル要素が生成し、かつ「width」プロパティが指定されていないボックスは、浮動化配置されたボックスの影響を受けない。中のインラインボックスは回り込みが起きるが、ブロックボックス自体には、回り込みが起きないのである。また、浮動化された要素の前にある要素は、それ以降にある要素の浮動化によって、なんの影響も受けないことにも注意させる。

ボックスが浮動化配置された場合、親要素の「height」の値の算出について注意させなければならない。html 要素を入れ子にした場合、通常は親要素の「height」の値を指定しない。その場合、親要素の「height」の値は初期値の「auto」のままになり、コンテンツの高さに応じて変化する。つまり、親要素は、コンテンツが内包されるように自動的に高さが計算されたレンダリングされる。しかし、フロート配置された子要素のボックスの高さは、親要素のボックスの

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/x
html1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>ボックスの浮動化</title>
<style type="text/css">
div {
  border:1px solid #000000;
  margin:1em;
}
.float {
  float:left;
  width:5em;
  height:10em;
}
.static {background:#ccc;}
</style>
</head>
<body>
<div>
  <div class="static">static<br />static<br />static<br />static</div>
  <div class="float">width:5em;<br />height:10em;<br />margin:1em;</div>
  <div class="static">static<br />static<br />static<br />static</div>
</div>
</body>
</html>
```

図 8　ボックスの浮動化の演習サンプル

高さの計算から除外される。その結果、浮動化されたボックスの高さによっては、図 9 のように、子要素のボックスが親要素のボックスからはみ出る場合が生じる。これが意図したレイアウトであるならばいいのだが、通常はそうではない。この問題は、親要素の「height」プロパティを指定すれば解決する。しかし、典型的な場当たり的解決法であり、好ましくない。親コンテンツの量が変更されたときに、「height」の値を指定し直す必要性が生じるからである。

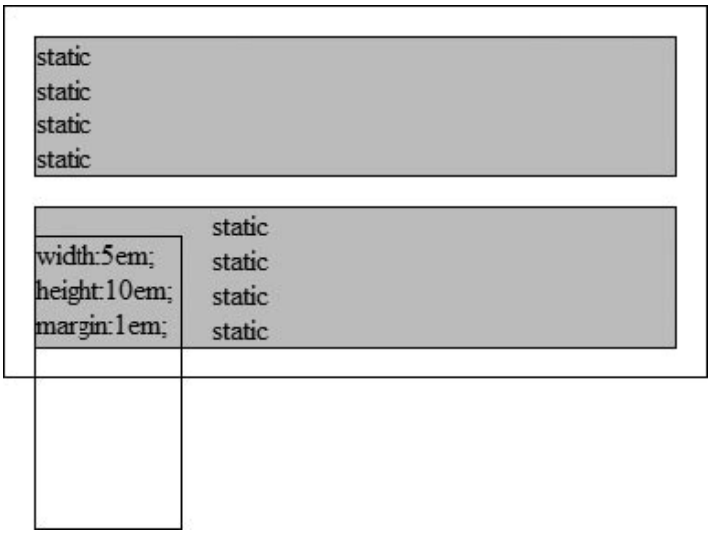

図 9　浮動化されたボックスとその上下のボックスの配置

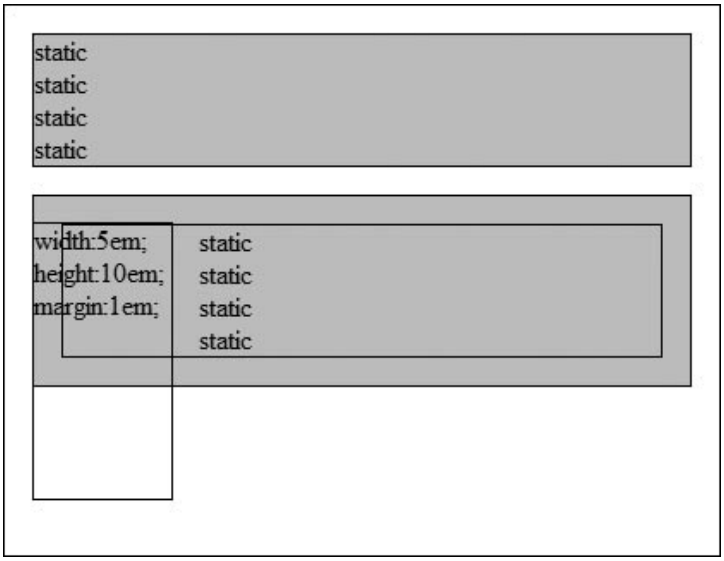

図 10　浮動化されたボックスに影響を受けないブロックレベル要素

極端な例を上げる。図 11 に示されるように、親要素のボックスの中に通常フローで配置された子要素が一つもない場合は、親要素のボックスの「height」の値は 0 になる。親要素のボックスに背景を設定していた場合などに問題となる。

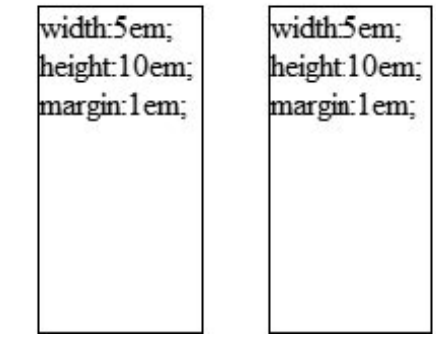

図 11　浮動化されたボックスとその上下のボックスの配置

この浮動化に関する副作用ともいえる現象は、初学者を大いに悩ませる壁となる。しかし、バグではなく CSS の仕様である。そして、頻出する現象だけに、これまで対処方法はいくつも発見されてきた。以下に代表的なものを上げる。

①親要素の CSS に「overflow:hidden;」という宣言を加える

②親要素の CSS に「float:left;」という宣言を加える

③親要素に対し、「clearfix」とよばれるテクニカルな CSS を書く

④親要素の中の最後の要素として「 」を加える

①は「overflow」プロパティを使う手法である。「hidden」を指定すると、親要素のボックスからはみ出た部分を隠すために、子要素のボックスの合計の高さが計算されることになる。この手法の利点は、論理的に納得しやすいことである。しかし、IE6 以下では、ブラウザの実装に問題があり機能しない。IE6 以下のブラウザで閲覧されることを想定するならば「zoom:1;」という宣言も加える必要がある[14]。②は親要素自体を通常フローから外してしまうという方法で、簡潔ではあるが、直感的に理解しにくい。また、親要素を浮動化することができない場合も考えられる。③は「:after」疑似要素クラスを使った方法で、浮動化の解除も同時に行える。プロが多用する問題解決法ではあるが、IE6 は疑似要素クラスをサポートしていないので、「zoom:1;」を加えることになる[15]。④は空の div 要素を作って、style 属性で浮動化解除の指定をする方法である。理解が容易であるが、デザインの崩れを HTML に手を加えて回避することになる。さらに、空要素ではないのに、コンテンツが空であり、好ましいマークアップとは言えない。文法的にはエラーである。「Another HTML-lint」では、重要度 1 の軽度のエラーとして検出される。

```
.clearfix:after {
    content:'';
    display:block;
    clear:both;
}
.clearfix {
    _zoom:1;
}
```

図 12　clearfix の標準的なコード

図 13 に示されているのは、上記の①～④のいずれかの方法を使った結果である。④はエラーであるので使いにくい。①～③の手法を比較すると、①と②は、下にフッターなどを加える場合、後続の要素に対し「clear:both;」の指定をする必要がある。③の手法は浮動化の解除を含んでいる分だけスマートである。ただし、下部の IE6 対策の部分が余分である。しかし、マイクロソフトによる WindowsXP のサポートは、2014 年で打ち切られる。すでに Google の Web サービス群においては、IE6 は動作保証されていない。WindowsXP のシェアの減少と共に、IE6 のシェアは減る。IE6 対策を念頭に置かなければならないのも、あと 5 年強であるといえる。以上のことを総合して考えると、③の手法のみを教えればよいといえる。

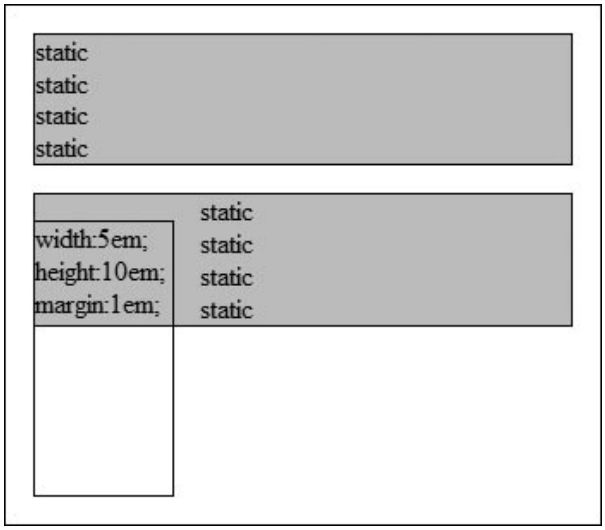

図 13　親要素のボックスの高さが確保された例

図 14 は、さらに、浮動化した要素に対して「margin-top」の値を指定し直した例である。子要素のボックス同士の重なりが回避されているが、浮動化の結果である後続要素のテキストの回り込みは解除されていない。上下の位置関係を変更するデザインは、「float」プロパティではなく「position:absolute;」プロパティで行うべきである。「top:10em;」「left:1em;」などと指定することにより、ブラウザの表示領域を基準とした任意の位置へのボックス配置が可能である。「float」プロパティは位置を自由に決めるためのプロパティではない。「『float』プロパティは、あくまで、左右にボックスを寄せる為のプロパティ」「インラインボックスの回り込みなどの現象は、ボックスを左右に寄せたことの副作用」と教えると誤解が少ない。

ボックスモデルとボックスの浮動化の演習は、できるだけ連続して実施するべきである。しかし、上記のような例を演習させると最低 2 時間は必要である。丁寧に説明しながら演習を行うとすると 3 時間は必要になる。

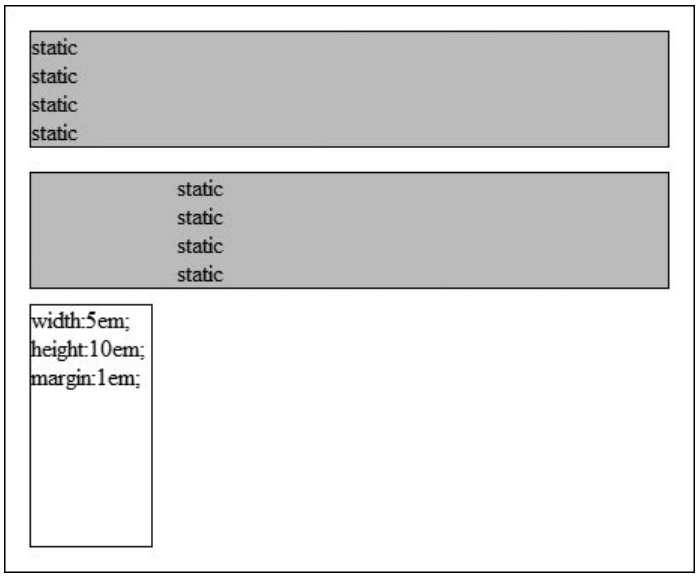

図 14　子要素同士の重なりを回避した例

3　HTML+CSS の総合演習

HTML で 1 時間、ボックスモデルとボックスの浮動化で 2 時間～ 3 時間の演習時間を見積もる必要がある。次のステップに必要な時間は、どの程度の CSS のデザインを学生に求めるかによってかかる時間は大幅に異なってくる。例えば、グローバルナビゲーションを例にとると、適切なマークアップにこだわらず、かつデザインにも凝らなければ、div 要素の中に a 要素を並べるだけである。「background」「list-style」「display」などの CSS のプロパティを教え、実務で使うデザインを適用できるレベルまで演習するとすれば、経験上、3 ～ 6 時間が必要である。特に background1 の使い方は多彩である。ロールオーバーイメージによるグローバルナビゲー

ションの装飾などを取り扱うと、それだけで2時間が必要になってしまう。いずれにせよ、ボックスモデルとボックスの浮動化の理解が得られた後は、ある程度、学生の努力と慣れに委ねることになる。

最後に、総合演習として、図15のような骨組みをもつページを、テーブルレイアウトとCSSレイアウトで、それぞれ1時間ずつ時間をとってデザインさせると、良い復習になる。また、CSSの「構造とデザインの分離」という優位性も実感させることができる。

図15　2段組レイアウトの演習サンプル

3-1　テーブルによる2段組みレイアウト

HTMLのみでレイアウトさせる場合、文書型宣言は「XHTML　1.0　Transitional」か「XHTML　4.01　Transitional」とする必要がある。CSSを使わない場合、table要素やtd要素の属性に「width」を用いる必要があるからである。これらの属性の使用は、Strict型では、文法エラーになる。すなわち非推奨属性である。テーブルレイアウトによる2段組のコードは図16に示される。セルの上部からコンテンツを配置するためには、「valign="top"」という属性指定が必要になる。また、table要素は、枠線は自動的にレンダリングされない。制作中は、「border="1"」で、td要素の枠を表示させておくとレイアウトが理解されやすい。枠を消したい場合には「border="0"」と指定させればよい。

```
<!DOCTYPE  html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml
1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>テーブルレイアウト２段組み</title>
</head>
<body>
<table border="0" width="600" bgcolor="#eeeeee">
<tr>
<td valign="top" witdh="80%" bgcolor="#ffffff">メイン･･･メイン</td>
<td valign="top" width="20%" bgcolor="#cccccc">サブ･･･サブ</td>
</tr>
</table>
</body>
</html>
```

図 16　テーブルによる２段組レイアウトのコーディング例

上記のような骨組みを作らせた後に、CSS によるデザインを加えると「ハイブリッドデザイン」と呼ばれるものになる。

3-2　CSS による２段組みレイアウト

HTML の div 要素に id 属性を付けさせる必要がある。ボックスの浮動化を理解していれば、テーブルレイアウトより容易である。

図 17 が CSS による 2 段組レイアウトのコードである。「構造とデザインの分離」が実現されているため、HTML のソースが非常に簡潔であることに着目させる。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD  XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>CSS2 段組みレイアウト</title>
<style type="text/css">
#container{width:600px; border: 2px solid #eee;}
#main{float:left; width:80%;}
#sub{float:right; width:20%; background:#ccc;}
</style>
</head>
<body>
<div id="container">
<div id="main">メイン･･･メイン</div>
<div id="sub">サブ･･･サブ</div>
</div>
</body>
</html>
```

図 17　CSS による 2 段組レイアウトのコーディング例

フッターを加える場合は、「 サブ・・・サブ 」という div 要素を、「id="sub"」の div 要素の下に入れる。CSS には、「#sub{clear:both;}」を加えて、浮動化の解除を行う[16]。

CSS のエラーチェックとしては、図 18 に示される W3C の Web サービスがよく用いられる。

図 18　CSS による 2 段組レイアウト

http://jigsaw.w3.org/css-validator/validator.html.ja

4　まとめ

近年、Web サイトはテキスト、画像、動画を表示するだけではなくなった。検索、通販、Web メールなどのインタラクティブなページでは、様々な Web サービスが提供されている。そのようなページでは、プログラミング言語が使われており、CGI、Java サーブレット、ASP、PHP などのコードが、動的に HTML や CSS を生成している。しかし、それらの HTML や CSS はプログラマによって記述されたものである。HTML と CSS をコーディングする知識は、Web サービスを作る場合も必要である。

半導体が「産業の米」と呼ばれるようになって久しい。WWW 誕生から 20 年経ち、クラウドコンピューティングが本格的に普及し始めた。クラウドコンピューティングのユーザインターフェースは、Web ブラウザに実装されることが多い。Web ブラウザの汎用性の高さを考慮すると、Web ブラウザ以外のユーザインターフェースは今後も考えにくい。HTML と CSS は「ICT 産業の米」とも呼べる存在になったといえる。

本論文では、HTML と CSS をできるだけ速く学ぶためのカリキュラムを提案した。内訳は、HTML に 1 時間、CSS のボックスモデルとボックスの浮動化に 2 時間～ 4 時間、CSS の重要なプロパティの使い方に 3 ～ 5 時間、総合演習に 2 時間である。合計では、8 時間～ 12 時間になる。大学の半期科目であると仮定すると、3 コマ～ 7 コマを DreamWeaver の使い方や自由課題の制作にあてることができる。なお、これらの見積もりはロジックで算出したわけではない。7 年間の授業の経験に基づいたものである。学生のレベルや、応用的な演習をどれくらい盛り込かによって、必要な時間は相当に変わる。しかし、本論文で提示したカリキュラムは実用性を重視したものであることは間違いない。主目的は、CSS のリファレンスをみながら、正統的な HTML+CSS のコードが書けるようにさせることにある。したがって、学問的には重要な CSS のカスケーディング、継承、プロパティがバッティングしたときの優先順位などが入っていない。しかし、これらの知識は、大規模なサイトを効率的に構築するためには必要である。したがって、大規模サイト制作のための速習カリキュラムの提案は将来課題とする。

（注）

（1）Google が全面的に採用したことによる

（2）XHTML で書かれた文書を XML 文書としても扱えることが HTML に対する大きなアドバンテージになるはずであった

（3）実際には、ほとんどこの形式が使われることがない

（4）div 要素を p 要素の代替として使用することは出来るが、段落は p 要素でマークアップするべきである。

（5）本格的にデザインを変更する場合には、ブラウザスタイルシートをリセットすることが多い

（6）厳密には全てのプロパティがボックスモデルと関係がある

（7）宣言と宣言の間の半角スペースはなくてもよい

（8）特にマージンを 0 にリセットするケースが多く、スタイルシートは「* {margin:0;}」と書かれる。

（9）font 関連のプロパティにも「font」というショートハンドプロパティが存在するが、値の指定順序を間違えると正常に動作しない。使いづらいので教えない方が無難である

（10）HTML の長さの属性値は、単位をつけてはならない。つけなくても、必ず「px」で解釈される

（11）仕様書では「positioned box」（位置決めされたボックス）と定義されている

（12）「パディング辺」と呼ばれる

（13）「ホームページビルダー」の「どこでも配置モード」は、絶対位置決めである

（14）「zoom」は IE に特有のプロパティであり、他のブラウザでは無視される

（15）「zoom」の前の「_」は、IE 以外で読み込ませないようにする為の記号であり、付けなくても問題ない

（16）正確には、ボックスの上マージンを増加させて、先行する浮動化されたボックスのボーダー下辺より、clear が指定されたボックスのボーダー上辺が下になるように配置させるプロパティ

参考文献

1）小松香爾（2009），「Web 制作のコアカリキュラム」，文京学院大学総合研究所経営論集，第 19 巻第 1 号