情報科教育法II　2003＃11

久野　靖*

2004.1.21

はじめに

前回「プログラミング」を取り上げましたが、プログラミングが苦手だという人が多かったわりには出席アンケートによると「情報」で取り上げることには肯定的な人が多かったです。取り上げると時間をとってしまうという意見もありましたが、何時間も使って本格的にやるのは普通教科「情報」としては…少なくとも「情報 B」でないなら、本筋ではないと思います。前回の JavaScript みたいなのを 2〜3 時間くらい体験してもらうのがいいんじゃないでしょうか。

あと、アンケートで「Java をやって嬉しかった」と書いた人がありますが、Java じゃなくて JavaScript ですってば。両方はまったく「別物」の言語なので混同しないでください (情報の教員としてはかなり恥ずかしい)。Java もちょっとだけ (次回とかに) 取り上げるかも知れません。今回はまだ JavaScript の続きをやりましょう。

その前に、まず「情報の教員であり続けるために」の話題を最初に少しします。あと、教員であれば生徒の評価は避けられない話題なので、「試験と評価」を取り上げることにしましょう。

1　「情報」の教員であり続けるために

1.1　なぜ「あり続けるために」か?

この話題は「情報科教育法」の 6 章でも取り上げられているが、それとは別の話題をここでは取り上げる。「情報科教育法」の方は各自読んでおいて欲しい。

ところで、プログラミングの話題にちょっとだけ戻ると、高校に行くと生徒の中にはプログラミング大好きな人は必ずいるので、「教師よりも生徒の方ができる」という状況は避けらない。別にプログラミングでなくても、ソフトの操作とかネットワークの知識とかグラフィクスの腕前とか、すべてにおいて同様。このあたりは、教科「情報」では避けられないところで、他の教科とははっきり隔たっている (「数学」の先生が生徒より数学ができないということはまずないでしょうね)。

では、教科「情報」の教師としては、何を自分の「ウリ」にしたらいいだろうか? その方向としては「メタな方向」「社会的側面」の 2 つがあると思う。以下で説明しよう。

1.2　メタな方向への発展

「メタな方向」というのは言い方を変えれば「1 つ上のレベルから見る」ということ。たとえば、ペイントソフトを使って画面の上でちょっと絵を描くことを考える。単に絵を描くだけなら、あなたがよほど絵がうまくない限り、生徒の中で絵がうまい人の方が上手なはずである。しかしその「具体的な操作」から上のレベルを考えてみたらどうだろう。

*筑波大学大学院経営システム科学専攻

レベル 0 ペイントソフトの消しゴムを選んで描いた絵を消すことができる — それはペイントソフトを使う人なら誰でも知っている

レベル 1 「消す」とはどういうことか? — 実はペイントソフトの「絵」は多数の「色のついた点」から成り立っていて、点の色は自由に変更できる。だから消しゴムの動いたところを「白」に変更すれば見た目「消した」ことになる。

レベル 2 しかし「消す」のと「白で塗る」のは常に同じとは限らない — なぜ「白」が「消した」ことになるかというと、元の地の色 (紙の色?) が「白」だから。実は多くのペイントソフトでは「前景色」(文字やペンの色) と「背景色」(地の色) をペアで持っていて、消すというのは背景色で塗ることになる。

レベル 3 「消す」代わりに「描いたのをやめる」こともできる— コンピュータ上でのすべての操作は「マウス等の入力」→「ソフトでそれを感知」→「それに対応して絵を変化」→「表示」という形で行われているので、どのような変化を起こしたかをすべて記録しておくことができる。その際、すべての変化に際して「もとに戻すためにはここをこう変えればよい」という形の情報も保存しておけば、現在の状態から過去に遡ってその情報を用いて「描いたものを取り消す」ことができる。別の方法として、時々絵を保存しておくことでその保存した状態まで戻ることはできるが、それだと「連続的に取り消して行く」のは難しい。

レベル 4 絵を多数の「レイヤー」に分けて描ける — 上のようにして 1 枚の「絵」を作るのは、人間にとっては大変。というのは、ある場所を後で直そうと思うとそこまで戻って、直して、やり直すのは大変だから。そこで、絵を多数の「透明なフィルム」に描いて、それを重ね合わせて最終的な絵にする、という機能を持つソフトが多い。これならば、絵を部分部分で別のフィルムに描き、失敗したフィルムは捨ててやり直すことができるから楽。

レベル 5 あるソフトが楽とか使い易いとは — まず、人間は「やることが不正確」「すぐ忘れたり間違えたりする」「面倒くさがり」といった特性を持っている。ソフトがそれをサポートしてくれることが大切。具体的には、…

1. 「筆やペンで描く」「消しゴム」「透明なフィルム」など人間にとってなじみのあるものになぞらえることで覚えやすくする (モデルを提供する)。
2. 失敗したら取り消して元に戻れる、失敗したフィルムは捨ててしまえるなど、不正確や間違いがあってもとり返せるようにする。
3. 「面倒くさい」のは決まり切ったことを繰り返しさせられるから。決まり切ったことはソフトにやらせてしまって、人間には「本当に面白い (創造性のある) ところ」を受け持ってもらえるようにする。

レベル 6 そもそも人間の「やることが不正確」「すぐ忘れたり間違えたりする」「面倒くさがり」は弱点であり是正すべきものなのか? — たとえば、コンピュータが非常に高価だったころは、人間が細心の注意を払って間違えないようにコンピュータを操作していた。しかし今ではコンピュータは廉価で人間の労働対価は高い。だから今ではコンピュータに最大限、人間のサポートをさせるべき。

レベル 7 しかし、ソフトが人間の間違いをサポートしてくれるのはいいことづくめとは限らない — 名古屋空港での中華航空機墜落事故では、コンピュータを間違って「自動復航モード」にした状態で着陸降下しようとしたため、人間が操縦桿を下向きにしてもコンピュータがそれを「補正」して上昇しようとし、最後に人間が着陸をあきらめて上昇しようとしたときに異常な' 急上昇になって失速墜落した。何が悪いと思う?

レベル 8 人間とコンピュータの関係について…まだまだ続く…

このように、ある事柄について考えるとき、その事柄の「枠組み」を与えている何かがあることに気付いて、その「枠組み」を問題にすることでレベルを上がっていくことができる。もちろん、上に行くほど抽象的なテーマになるので、どこまで行くかは生徒の状況に応じて判断するが、少なくともこういう形で枠組みを考えさせることは大切だし、その部分については教員がリードしていくことができる (はずですね???)。

1.3 社会的側面

上の例でも分かるように、メタなレベルを遡って行くとしまいには「人間とコンピュータ」のような社会的側面に行きつく。それでなくても、「ネット上でどのように行動するべきか」「電子メールではどういうことに気をつけるべきか」などの部分は社会的側面。この授業でも「情報倫理」「著作権」をはじめ、社会的側面については多く取り上げて来ている。

人間が社会的側面について学ぶのは主として自分の社会的体験によっている。しかし、高校生はまだ「学校の中」が主な社会であり、社会的体験が不足している。しかし「知らない」ままでは済まされない…ではどうするかというと、最終的には「教員の社会的体験を生徒に語ることで疑似体験してもらい、それを通じて納得を積み上げる」ことが必要だと思う (それなしに「これはよい」「あれは悪い」という知識レベルでの詰め込みをしても身にならないし、具体的な事例として学ばなかったことが判断できない)。

だから、教員は自分の中に十分な「社会的体験」と「それに対する自分としての考え方の姿勢」をストックしていて、折に触れて生徒にそれを伝達するように努めるべきだと思う。逆に言えば、教員になったら学校に閉じ籠って学校だけが自分の世界、だとすぐストックが底をついてしまう。意識してさまざまな社会的体験を持つように心がける必要があると思う (遊び回れとは言っていないので念のため)。

1.4 情報収集/ストックとフロー

それはそれとして、コンピュータ関係のことがらはものすごい勢いで変化するので、ちょっと勉強を怠っているとすぐ「時代後れ」になる。たとえ自分の持っている「判断方法」が正しくてもデータが古ければ「間違い」の結論に到達してしまうので、教科「情報」の教員であればそれこそ「情報収集」は怠らないようにすべきである。

このとき、情報を単に「積み上げて」頭に入力しても身につかない。自分なりのさまざまな情報の関連しかた (全体的な「構図」) が頭の中にあって、そのそれぞれの場所に新しい情報をくっつけ、古くなった情報は「倉庫」にしまって (忘れる必要はない…「昔話」も結構教える役に立つものですから)、絶えず自分の「絵」を新しい状態に保つようにする。

言い替えると、自分の「絵」をまず作り出すこと (最初のストック)、そしてその「絵」を新しい情報で絶えず更新していくこと (その後のフロー) が両方とも必要、ということですね。皆様ももう大学生の終盤なわけでしょうから、「最初のストック」が十分じゃないと自分で思う人は心がけて「絵」を完成させるように努めてください。

2 模擬授業

今回の模擬授業は「情報 C」p30〜31 だそうです。担当の方はよろしく。

3 教科「情報」における試験と評価

教科「情報」も学校の教科である以上、評価を行って成績をつけなければならないのは当然である。しかし、「情報」はこれまでの教科にない特性をいろいろ持つため、その評価方法について、さまざまな流言 (多くは誤解に基づくもの) が飛び交っている。

たとえば、次のような意見のうちどれが正しくてどれが間違っていると思うか?

(1) 「情報」ではコンピュータの操作ができるかどうかを見なければならないので、試験はコンピュータの前での実技試験が中心となる。(体育モデル)

(2) 「情報」であっても普通の教科と同様、紙を使ったペーパーテストが主流となる (製作実習などは作品や発表を評価することもあるが)。(通常教科モデル)

(3) 「情報」の評価は Web ページやプレゼンテーションなど製作物の評価やパフォーマンスの評価が中心となる。(美術・音楽モデル)

(4) 「情報」の試験勉強にはコンピュータが必要なので、生徒の親にはコンピュータを買ってもらうようお願いする。また自宅にコンピュータのない生徒に対しては試験前には公平にコンピュータルームで勉強できるようにくじ引きで割り振りを行うとよい。

(5) 「情報」でも試験の勉強は教科書や配付資料・問題集が中心なので、生徒が試験前に「試験勉強」と偽ってネットで遊ばないように、親には「試験 2 週間前からはコンピュータ使用を禁止してください」等の通達を出しておくのがよい。

(6) 「情報」でよい成績を納める生徒は実際にコンピュータを使った経験を沢山積んでいる必要はない。むしろ教科書などをよく理解しているかどうかを評価すればよい。

(7) 「情報」でペーパーテストによる評価をするとしても、その問題を工夫して、実際にコンピュータを操作して経験を積んでいることが評価の上で現れてくるような問題を作ることが望まれる。

(8) 「情報」のペーパーテストを作るとなると、必然的にコンピュータやネットワークなどに関係するさまざまな概念の知識を問う問題が中心となるだろう。

(9) 「情報」のペーパーテストであっても、できるだけ「考える力」「計算する力」を見るような問題を盛り込んで、暗記だけでは済ませられない方向に行くのがよいだろう。

どう思いましたか? 自分としては、「情報」の評価はペーパーテスト中心で構わない (むしろペーパーテストによる評価になじむ内容を多く含んでいる)、作品やパフォーマンスの評価も一定量含んでいるものになるだろう、つまり「一般教科モデル」と「美術・音楽モデル」をたとえば 3:1 くらいの比率で混ぜたものになるのでは、と思っている。

問題は、ペーパーテストを安易に作るとどうしても穴埋めなど知識を問うだけの問題になってしまい、それではせっかくコンピュータを使ってさまざまに体験を積んだことが評価の上でプラスに働かない、したがって成績のためには実習をそこそこにして問題集ばかり勉強する、という方向に生徒が行ってしまう、という点で困ると考える。これを打破するには、同じペーパーテストであってもできるだけ「問題をさまざまな側面から分析検討することで解ける」もの、また「実際にコンピュータで実習した体験があると分かるようになる」ものを増やすことだと思う。まあこう書くほど簡単ではないと思うが、、、、このあたりはまた次回にも取り上げる。

ところで、「情報」に関するセンター試験があるのはご存じだろうか? 実は普通教科「情報」ではないのだが、「情報関係基礎」という科目が 10 年近く前から存在している。この科目は工業高校・商業高校などで「情報処理」(「情報」ではない!) を学んだ学生が対象で、数学の中の 1 科目として選択できる。情報処理なのでプログラミングに関する問題がみっりあり、その点で「情報」ではないが、例年「大問 1」は教科「情報」であってもおかしくないような問題が出されているので参考になる。

「情報」そのもののセンター試験はどうなのだろうか? 実は大学入試センターでは「情報」をセンター試験に入れるか否かをまだ「検討」している段階である。もしセンター試験に入らないとなると、「情報」がお遊び科目になってしまい、進学校ではまじめに授業をやってもらえない (数学等の受験科目に振り返られて消されてしまう) 恐れがあるので、自分としてはぜひセンター試験でやって欲しいのだが…

ともかく、どんなものか見てみないとお話にならないので今回は先日あったばかりの「情報関係基礎」の問題をコピーしてきました。ぜひ最初の方を解いてみてください。

4 JavaScript による Web ページ内容の操作

4.1 Document Object Model (DOM)

ここまで HTML によるページの作り方をいろいろ学び、そして JavaScript を使うと HTML 中の「フォーム」の値を取り出して操作できることも学んだ。しかしもっと恐ろしいことに! 実は JavaScript を使えばページ内の**任意の**内容を自由勝手に変更することができる。

実は、HTML によって記述されたページの内容は木構造のデータとしてブラウザ内部に蓄えられている。木構造のノード (節) は、HTML の要素 (body とか p とか h1 とかすべて) に対応している。そして、それぞれのノードは次のようなフィールドを持っている。

- `parentNode` — 親の (上の) ノードを指す
- `nextSibling`、`previousSibling` — 「次隣」「前隣」のノードを指す

また、各ノード N は次のようなメソッド (操作) を持っている。

- N`.removeChild(`ノード`)` — 子ノードを取り除く
- N`.appendChild(`ノード`)` — ノードを一番最後の子供として追加する
- N`.insertBefore(`ノード`,` $N1$`)` — ノードを既にある子ノード $N1$ の直前の子供として追加する
- N`.cloneNode(true)` — N 以下の木構造をそっくりコピーした木構造を作って返す

このような、ドキュメント内部の構造とその操作を定めた標準を DOM(Document Object Model) と呼ぶ。標準化されているので、IE6 と Netscape6/7 で共通に使える (少し古いブラウザはダメ)。

これを利用して、箇条書の項目を「ぐるぐる回し」するサンプルを示そう。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>JavaScript Sample</title>
<script type="text/javascript">
function chg(e) {
  var parent = e.parentNode, prev = e.previousSibling;
  parent.removeChild(e);
  if(prev) parent.insertBefore(e, prev); else parent.appendChild(e);
}
</script>
</head>
<body>
<ul>
<li>これは犬です。</li>
<li>これは猫です。</li>
<li>これはペンです。</li>
<li onmouseover="chg(this)">これは電車です。</li></ul>
</body>
</html>
```

このページの「これは電車です」という li 要素の上にマウスが来ると、この li 要素 (`this`) を引数として関数 `chg()` が呼び出される。`chg()` の中では次のことをやっている。

- 変数 `parent`、`prev` に親ノードと 1 つ前隣のノードを入れる。

- 自分を親ノードの子供からはずす。
- 前隣があれば、自分をその前隣の 1 つ前のノードとして挿入し直す。前隣がなければ、自分を最後の子供として挿入し直す。

これで、「これは電車です」の上にマウスポインタが載ったとたん、このノードは 1 つ上に向かって「逃げる」ことになる。結構面白いでしょう?

演習 1 このプログラム (`/home/guests/kuno/work/js5.html`) をコピーしてきてそのまま動かせ。動いたら次のように変更してみよ。

a. もっと箇条書の項目を増やして同じに動作することを確認。
b. 他の項目いくつかも同様に「逃げる」ようにする。(ヒント: `onmouseover` の指定を同様につければよい。)
c. 移動するのではなく、自分のコピーが挿入されるようにする。(ヒント:「`parent.removeChild(e);`」を「`e = e.cloneNode(true);`」に取り換える。)

4.2 DOM によるスタイル操作

DOM を使ってドキュメントの内容をぐちゃくちゃにいじくるのは過激なので、普通はもっとおだやかにスタイル (見え方) を変更するくらいの使い方が普通。スタイルを変更するのは非常に簡単で、各ノード (つまり HTML 要素) に対して `.style` という属性が用意されていて、そのさらに属性として CSS のプロパティ名と同じものを指定して値を入れればよい。ただし、「-」は引き算になってしまうので削除し、かわりにその次の文字を大文字にする。たとえば「`background-color`」という CSS プロパティは DOM 側から使うときは「`backgroundColor`」になる。復習も兼ねて代表的なものをあげておく。

- N`.style.color` — 文字の色
- N`.style.backgroundColor` — 地 (背景) の色
- N`.style.backgroundImage` — 背景画像
- N`.style.border` — 枠のスタイル
- N`.style.textDecoration` — 文字飾り (点滅、下線など)
- N`.style.left` — 位置指定 (X 座標)
- N`.style.top` — 位置指定 (Y 座標)

ではこれらを使ってスタイルを変更する例を見てみよう。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>JavaScript Sample</title>
</head>
<script type="text/javascript">
function move1(e) {
  e.style.left = (400*Math.random()) + 'px';
  e.style.top = (400*Math.random()) + 'px';
}
</script>
<body>
```

```
<p onmouseover="this.style.backgroundColor = 'red'"
onmouseout="this.style.backgroundColor = ''">これはペンです。</p>
<p onmouseover="this.style.border = 'blue solid 4px'"
onmouseout="this.style.border = 'none'">これはミカンです。</p>
<div style="position:absolute" onmouseover="move1(this)">亜</div>
</body>
</html>
```

「これはペンです」はマウスが上にのると背景が赤くなり、マウスがどくと元に戻る。「これはミカンです」は同様だが枠がついたり消えたりする。なお、元に戻すとき「''」を入れる場合と「'none'」を入れる場合とがあるけど説明が難しいのでまあ適当に試して使ってください。3 番目に、「亜」という文字の上にマウスがのると、その文字が別の位置にワープする。これは別に用意した関数を読んでその中で乱数を使って計算している (Math.random() は 0 以上 1 未満の一様乱数を返す)。

演習 2 このプログラム (/home/guests/kuno/work/js6.html) をコピーしてきてそのまま動かせ。動いたら次のように変更してみよ。

a. 変更の内容として、文字の色とか文字飾り (underline とか blink とか) もやってみる。
b. マウスが載った時でなくクリックしたときに変化するようにしてみる。(ヒント: onmouseover の代わりに onclick で同様に指定する。)
c. ランダムに逃げる文字を増やしてみる。
d. 逃げるたびに色もランダムに変化させる。(ヒント: 0〜255 の整数の一様乱数は乱数と整数への切捨て Math.floor() を利用して、たとえば

```
'rgb(' + Math.floor(256*Math.random()) + ',0,0)'
```

などとすればできます。上の場合は赤だけランダムだけど。)

なお、補足だけどインライン画像を差し替える場合 (いわゆる画像のロールオーバー) は CSS ではなく、画像オブジェクトの src 属性として格納されているファイル名を取り換えればよい。たとえば次のようにする。

```
<img src="a.gif" onmouseover="this.src = 'b.gif'"
onmouseout="this.src = 'a.gif'>
```

もちろん、背景画像を取り換える場合は CSS を使う。

4.3 イベントハンドラとイベントドリブンプログラム

ここまで散々使ってきたが、onmouseover みたいに「〜が起きたらこういう動作をする」というスタイルで動作するプログラムのことを「イベントドリブン」という。ユーザの操作に感応する (対話的な) プログラムはだいたいイベントドリブンなプログラムになる。

普通の言語だけで (たとえば C とか C++とか Java) イベントドリブンなプログラムを作るのは、「〜が起きたとき」に動作するべき内容を関数などの形で用意して、その関数を「登録」する必要がある。このような関数を、「イベントを処理する」という意味で「イベントハンドラ」と呼ぶ。イベントハンドラを分けて用意したり登録したりするのは結構繁雑で大変。

これに対し、HTML+JavaScript であれば、HTML 要素の属性としてイベントハンドラを書いてしまえるので、このあたりがずっと楽。この点でも JavaScript の簡便さ、楽ちんさはありがたい。具体的にどういうイベントハンドラがあるかを説明しておく。

- onload、onunload — ページが表示された時と、表示を終る (よそのページへいく) 時に呼び出されるハンドラ。

- onclick — マウスクリックがあった時に呼び出されるハンドラ。
- onmouseover、onmouseout — マウスポインタが載った時/出た時に呼び出されるハンドラ。
- onfocus、onblur — その要素にフォーカスがあたった時/はずれた時に呼び出されるハンドラ。

4.4 特定ノードの取得

ここまでの例ではすべて、ハンドラの中では this を指定して「この要素を変更する」という形で書いてあったが、遠隔操作みたいに「この上にのったらあっちを変更する」という場合もある。そのときは this では指定できないので、次のようにする。

- 指定したい (操作したい) 要素の HTML 開始タグに「id="名前"」という属性を指定する。名前は英数字の並びならなんでもいいが、ただし数字で始まらず、他の要素の id と重ならないこと。
- その要素を参照したければ「document.getElementById('名前')」で参照できる。

演習 3 演習 2 のプログラムを直して、マウスポインタが載った/ 外れたときに全く離れた場所の何かが変化するようにしてみよ。

A 次回までの追加課題: 試験問題を作る

「試験と評価」については次回もやりますが、そのネタとして次回までの追加課題として試験問題を作ってみていただきます。要件としては次の通り。

1. 試験範囲は「情報 C」教科書の序章を除く章から 1 節 (それで広すぎる場合はその一部の単元に限定してもよい) を選ぶこと。
2. 問題の形式は「○×」「穴埋め」「選択問題」「計算問題」「自由記述」など自由だが、採点基準を明示する必要がある。自由記述の場合、回答から曖昧さなく点数がつけられるように工夫すること。
3. 今回講義で述べたように、知識のみを問う問題はできるだけ避けて考える力やコンピュータを使った実習経験が活かせる問題をめざすこと。

以上の要件を満たすようにして、次の形で作ってください。

5. 問題は A4 版の紙 1 枚とし、冒頭に「試験問題: 問題名」(問題名は適当につける)「日付」「学籍番号」「作成者氏名」を記入する。
6. その下に「回答日付」「回答者氏名」の欄を用意する。
7. 問題数などは自由だが A4 版 1 枚におさめ、記入欄も取る。問題番号を適当につける。回答時間は 20 分とする (だからほんの少ししか出題できないと思ってください)。
8. 穴埋め回答欄は「[ ]」などの形で示す。

あと、正解・採点基準も作ってください。

9. これも A4 版 1 枚とし、冒頭に「正解・解説: 問題名」「日付」「学籍番号」「作成者氏名」を記入する。
10. 各問ごとに正解を曖昧さなく明記する。自由記述の場合は採点基準がはっきりするように。
11. 各問ごとに配点を明記し、合計点数 (満点点数) も明記する。

次回授業で実際にやってもらうつもりですので、次のように用意してください。

12. 試験問題は 3 枚コピー。正解の方は 2 枚コピー。次回授業時に忘れずに持参すること。