Part 1. **html 序論**

1. HTML から始めよう

1.1. **html から始めよう.** ホームページなどの web ページは, html という言語を基本として, javascript, css などの言語と組み合わせて作成されている. html は web ページの本体, javascript は web ページを動的に扱うために, css は web ページの見た目を修正するために用いる. 本稿ではこのうち html と javascript を扱う. この二つがあれば, 実用上, 最低限の web ページが作成できるからである.

まず, windows であれば「メモ帳」を開こう[1]. 次に, メモ帳に

```
<html>
  <head>
    <meta charset = "utf-8">
    <meta http-equiv="Pragma" content="no-cache">
    <style> body { font-family: monospace,serif; } </style>
    <title>あああ</title>
  </head>
  <body>
    いいい    <br>
    ううう

    えええ
    <!--
      おおお
    -->
    <script>
      document.write("かかか");
    </script>
  </body>
</html>
```

と書く. 半角と全角の違いに注意すること. いま書いたものを**ソースコード**と呼ぶ. さてこれを, 拡張子を「.html」にして保存する. 保存する場所はどこでもよいが, 今後のことを考えて, 使いやすい場所に保存することをお勧めする. ここでは, desktop 上のフォルダー「files」を作り, その中に「empty.html」として保存したとしよう. フォルダー「files」の「empty.html」をダブルクリックすると, ブラウザ上で「empty.html」が開く. 小さな web ページの出来上がりである.

さて, ソースコードに戻ろう. `<head>`と`</head>`の間を**ヘッダー**と呼び, `<body>`と`</body>`の間を**ボディー**と呼ぶ. 開いた「empty.html」を確認すると, タブのところに「あああ」というタイトルが, そしてページそのものには「いいい (改行) ううううえええかかか」と記されていることが分かる. お察しの通り, ヘッダーに書かれた「`<title>`あああ`</title>`」がタイトルを指定しており, ボディーに書かれた内容がページそのものに表示されていることが分かる. web ページのプログラミングにおいては, もっぱらボディーにプログラムを書き込んでいくことになる.

ソースコードは命令の塊で, ブラウザはその命令を読み取り, その命令通り処理する. プログラムを理解するには, まず命令と処理を理解する必要がある.

[1] メモ帳は, ディスプレイ左下のスタートをクリックして, 「w」の項目の中の「windows アクセサリ」の中にある.

1.2. **タグ.** html のソースコードをよく見ると, <○○>という文字列が出現している. これらを**タグ**と呼ぶ. ここまでで現れたタグは

`<html>`, `<head>`, `<meta ○○>`, `<style>`, `</style>`, `<title>`, `</title>`, `</head>`, `<body>`, `<br>`, `<!-- ○○ -->`, `<script>`, `</script>`, `</body>`, `</html>`

の計 15 個である. これらのタグは html における命令であり, それぞれブラウザ上である内容を実行している. 本稿は, 主として javascript を学習することを目標としている. 実はこの 15 個がこの講義で登場する html のタグの全てである. これ以降は, javascript について学習することになるが, html の学習の最後に, いくつかタグについて知っておこう.

1.3. **改行タグ.** `<br>` を**改行タグ** *(break* **タグ***)* と呼ぶ. 文字通り, 改行する命令である. 「いいい」と「ううう」の間で改行されているのはこの改行タグの所為である. なお, ソースコード上で改行されていても改行タグが記されていなければ改行されないことに注意しよう.

補足 1. 「ううううえええ」の部分を見よ.

1.4. **コメントアウトタグ.** `<!--○○-->` を**コメントアウトタグ**と呼ぶ. これは「○○に書かれたことは実行するな」という意味である. html や一般のプログラムにおいて, ソースコードに書かれたことはすべて実行の対象となる. しかし, これは人間にとっていささか窮屈である. そこで, どんなプログラミング言語にもコメントアウトという機能がついており, html ではそれがコメントアウトタグである. ここに書かれたことをブラウザは無視する.

1.5. **script タグ.** `<script>` と `</script>` を *script* **タグ**と呼ぶ. この `<script>` と `</script>` に囲まれた部分を *script* **環境**と呼び[2], ここが javascript の主戦場である. ここは javascript というプログラミング言語で書かなければならい. じつはここに書かれている `document.write(○○)` というのが javascript の命令である. script 環境の内部では javascript で書き html で書いてはならない. 逆に, script 環境の外部では html で書き javascript で書いてはならない. html ファイルはこのように html と javascript という二つの異なる言語で書かれている[3].

[2] いくつかのタグはこのように開始と終了の対をなす. 例えば, `<html>` と `</html>`, `<head>` と `</head>`, `<title>` と `</title>`, `<style>` と `</style>`, `<body>` と `</body>` なども環境である. これに対して, `<meta ○○>` や `<br>` や `<!--○○-->` は単独で用いられる.

[3] 正確に言うと, style 環境内部では css で書き html で書いてはならない. この意味では, html ファイルは html と javascript と css という三つの異なる言語で書かれている.

Part 2. javascript

ここからが javascript である.

2. 出力

ここでは, 出力として,

- document への出力
- alert による出力

を扱う. これ以外にも様々な出力があるが発展的なのでここでは扱わない.

2.1. **document への出力.** 概ねページへの文字の表示のことと思えばよい. 例えば,

```
document.write("かかか");
```

とすると, ページに「かかか」と表示される. ただし, 正確に言えば, 「html のソースコードに『かかか』と書け」という意味である. 例えば,

```
document.write("<br>");
```

とすると, 「html のソースコードに『
』と書け」という意味になるから, ここで改行されることになる.

補足 2. では, ページに「
」と表示したい場合はどうすればよいか. 対応策はふたつある.

(1)
```
document.write("＜ br ＞");
```
と書く. これは全角の「＜」「＞」の利用である. 全角文字と半角文字は別の文字なので, タグとは解釈されない. とは言え, 半角と全角は見た目が異なるので, これでは不満足かもしれない. より適切には, 次のように書く.

(2)
```
document.write("\<br\>");
```
と書く. この \ を**エスケープ**と呼ぶ. 半角の <, > は html のタグと解釈されるので, そのまま表示することができない. この解釈から逃れるためにエスケープ文字 \<, \> を用いる.

2.2. **alert による出力.**

```
alert("かかか");
```

とすれば, アラートで「かかか」と表示される. この際に「OK ボタン」も表示され, ユーザーが「OK ボタン」をクリックすると, ページに戻る.

補足 3. このことを利用すると, ユーザーに待機させることができる.

2.3. **コメントアウト.** html でのコメントアウトの仕方についてはすでに述べたので, ここでは, javascript でのコメントアウトの仕方を紹介する.

```
// この一行はコメントアウトされる.
```

とすると, コメントアウトされる. html と javascript は別の言語である. コメントアウトの仕方も異なることに注意しよう. また,

```
/*
  ここは
  まとめて
  コメントアウト
  される.
*/
```

とすると, まとめてコメントアウトできる.

3. 変数

3.1. 変数の宣言と変数への代入.

```
let x = 3;
```

と書くと, 変数「x」が**宣言**され, x に数値 3 が**代入**される. 実際,

```
let x = 3;
document.write(x);
```

とすれば, 「3」と表示される. これを

```
let x;
x = 3;
document.write(x);
```

と書いてもよい. ここでは, 「let x;」が宣言を意味し, 「x=3;」が代入を意味する. 「let x=3;」は宣言と代入をまとめて行なう記法である.

補足 4. プログラムと数学は極めて親和性が高いが, 等号については意味が異なる. 数学において等号=は等しいこと「=」を意味するが, プログラムにおいては等号=は代入「:=」を意味する.

代入についてもう少し正確に述べると, 「○○=△△」という書式では, ○○は変数でなければならない. 例えば,

```
3 = 3;
```

や

```
3 = 2;
```

とすると, エラーが生じる. これは, 左辺が変数でないからである. 等号の左辺は変数でなければならない. その前提の下で, この式は, 「△△の値を変数○○に代入する」という意味である.

変数には, 数値だけでなく, 文字列を代入することもできる. 例えば,

```
let y;
y = "こんにちは";
document.write(y);
```

とすると, 変数「y」が宣言され, y に文字列「こんにちは」が代入される.

補足 5. 変数には様々なものを格納することができるが, 代表的なものは「数値」と「文字列」である.

3.2. 変数の再代入.

```
let x;
x = 3;
document.write(x);
document.write("<br>");
x = 5;
document.write(x);
```

とすると, 「3(改行)5」と表示される. このように変数へは再代入が可能である.

3.3. **変数の同一スコープ内での再宣言の禁止.** しかし,

```
let x;
x = 3;
document.write(x);
document.write("<br>");
let x;
x = 5;
document.write(x);
```

とすると, エラーが生じる. これは変数 x がひとつのスコープ内[4)]で再宣言されているからである. ひとつの変数は一つのスコープ内では一度だけしか宣言できない[5)].

3.4. **変数とスコープ.** 本小節の内容は少々難しいので, 初読の際は無視してよい.

{と}で囲まれた部分を**スコープ**と呼ぶ. 括弧は自由に入れ子にすることができるが, 括弧の開きと閉じは正しく対応しなければならい[6)]. あるスコープ内で宣言した変数は, そのスコープ外では利用できない. 例えば,

```
{
  let x = 3;
  document.write(x);
  document.write("<br>");
}
document.write(x);
```

とすると, エラーが生じる. これは, 1 行目で宣言された変数 x は, それが属すスコープ (0 行目〜4 行目) の中でしか有効でないからである. したがって, 5 行目においては未宣言の変数 x を利用された javascript はエラーを吐くのだ.

一方で, あるスコープで宣言された変数は, その子孫のスコープすべてで利用可能である. 例えば,

```
{
  let x;
  x = 3;
  document.write(x);
  document.write("<br>");
  {
    x = 5;
    document.write(x);
    document.write("<br>");
  }
  document.write(x);
}
```

とすれば「3(改行)5(改行)5」と表示される. このプログラムでは, 子スコープ 5 行目の代入によって, 親スコープで宣言された変数 x に再代入が行なわれている. この再代入によって, 親スコープ 9 行目でも変数 x の値が変更されていることに注意しよう.

[4)] スコープについては後述.

[5)] 数学と同じである.

[6)] 数学と同じである.

3.5. **変数名.** 変数名としては

- 半角英字小文字:a〜z
- 半角英字大文字:A〜Z
- 半角数字　　:0〜9
- アンダースコア:_
- ドルマーク　　:$

の計 64 種[7]を 1 字以上連続させたものが利用できる. 大文字と小文字は区別される. ただし,

- 半角数字から始まる物
- 予約語

は利用できない. 予約語については, 例えば

`https://www.javadrive.jp/javascript/ini/index5.html`

を参照せよ.

補足 6. 変数名はルールを守っている限りどのような名前を用いてもよい. プログラムを他人に見せる予定がないならば, もっとも重要なことは自分が間違わないような変数名を使うことである. すべての予約語は英単語なので, 例えば, 日本語単語のローマ字表記などを利用すれば, 予約語と衝突することはない. とは言え, ローマ字表記は読みにくいうえに表記の一意性がない[8]ので, 記述ミスが生じやすい.

しかし, もし他人に見せることを想定するのであれば, 可読性を考慮した変数名を使うべきである. このような観点からよく使われるマナーとして, 以下の 3 つを挙げておく:

- キャメルケース: 2 単語目以降の先頭を大文字にする.

```
let userName="nakada";
```

- パスカルケース: すべての単語の先頭を大文字にする.

```
let UserName="nakada";
```

- スネークケース: すべて小文字で, 単語と単語の間をアンダースコアで繋ぐ.

```
let user_name="nakada";
```

[7]実際には unicode 文字も利用できる.

[8]`zettaichi` と `zettaiti` など.

4. 演算

javascript においては, 予め定義されている演算がある.

4.1. **数値に対する演算.** javascript では以下のような演算が利用できる:

```
let a=13;
let b=3;
document.write(a+b); // 和 16
document.write("<br>");
document.write(-b); // 反数 -3
document.write("<br>");
document.write(a-b); // 差 10
document.write("<br>");
document.write(a*b); // 積 39
document.write("<br>");
document.write(a/b); // 商 4.333333333333333
document.write("<br>");
document.write(a%b); // 余り 1
document.write("<br>");
document.write(a**b); // 冪乗 2197
```

特に, 余りについてはその挙動をよく確認しよう. これはプログラミング言語によって挙動が異なる. javascript では,

```
document.write(13%3); // 1
document.write("<br>");
document.write(-13%3); // -1
document.write("<br>");
document.write(13%-3); // 1
document.write("<br>");
document.write(-13%-3); // -1
```

となる. つまり, 環論の言葉で言えば, javascript における余りは最小非負剰余でも最小絶対剰余でもないことになる.

4.2. **代入再論.** 代入の書式「○○=△△」は「△△の値を変数○○に代入する」ことであると言った. これは「△△を変数○○に代入する」ではない. 「の値」とわざわざ言及しているところに意味がある. 例えば,

```
let a = 13;
a = a+3;
document.write(a); // 16
```

を見よ. 1 行目において,「a+3」の値は 16 なので, 1 行目は「16 を変数 a に代入する」という意味である. 結果的に 1 行目は「変数 a の値を 3 増やす」という意味になる.

4.3. **演算して代入.** 上の様な手続は頻繁に用いることがあるので, 略記法がある.

```
let a;
a = 13;
a += 3;
document.write(a); // 16
document.write("<br>");
a = 13;
a -= 3;
document.write(a); // 10
document.write("<br>");
a = 13;
a *= 3;
document.write(a); // 39
document.write("<br>");
a = 13;
a /= 3;
document.write(a); // 4.333333333333333
document.write("<br>");
a = 13;
a %= 3;
document.write(a); // 1
document.write("<br>");
a = 13;
a **= 3;
document.write(a); // 2197
```

4.4. **インクリメントとデクリメント.** 「a=a+1」と「a=a-1」は頻繁に用いるので, さらなる略記が用意されている:

```
let a;
a = 13;
a++;
document.write(a); // 14
document.write("<br>");
a = 13;
++a;
document.write(a); // 14
document.write("<br>");
a = 13;
a--;
document.write(a); // 12
document.write("<br>");
a = 13;
--a;
document.write(a); // 12
```

「a++」と「++a」を**インクリメント**, 「a--」と「--a」を**デクリメント**と呼ぶ.

4.5. **整数値に対する演算.** より発展的な演算として以下のものを紹介するが, 初読の際は無視してよい.

```
let a=17;
let b=3;
document.write(a|b); // ビット和 19
document.write("<br>");
document.write(~b); // ビット反転 -4
document.write("<br>");
document.write(a&b); // ビット積 1
document.write("<br>");
document.write(a^b); // ビット排他和 18
document.write("<br>");
document.write(a<<b); // ビット左シフト 136
document.write("<br>");
document.write(a>>b); // ビット右シフト 2
```

これらは a, b が整数値でないと正しく機能しない. これらの意味は a, b を二進法表示すると分かるだろう. これらを中々使う場面もないだろうが, こんな演算もあるのか, と思ってもらえればよい.

4.6. **文字列に対する演算.**

```
␣␣let␣a␣=␣"Hello";
␣␣let␣b␣=␣"world";
␣␣document.write(a+b); // 連結 Helloworld
␣␣document.write("<br>");
␣␣document.write(a+""+b); // Helloworld
␣␣document.write("<br>");
␣␣document.write(a+"␣"+b); // Hello␣world
␣␣document.write("<br>");
␣␣document.write(a+"␣␣"+b); // Hello␣world
```

なお, ""は**空文字列**を意味する. (※) 強調するために半角空白を␣と表示してある.

補足 7. 代数の言葉で言えば, 文字列の全体は, 連結を演算として空文字列を単位元とする非可換半群をなす.

また, html において, 連続する半角空白は半角空白 1 つ分と等価である[9].

4.7. **数値と文字列が混ざったら.** 演算 + は数値と数値に対しても文字列と文字列に対しても用いられる. しかし, 数値と文字列に対して, あるいは, 文字列と数値に対して用いる場合は, 文字列に変換されて扱われる.

```
let a = "Hello";
let b = 3;
document.write(a+b); // Hello3
document.write("<br>");
document.write(b+a); // 3Hello
```

[9] 多くのプログラミング言語で連続する半角空白は半角空白 1 つ分と等価である.

4.8. **数と数字.** 基礎代数でも述べた通り, 数と数字は明確に異なる概念である. これはプログラミングにおいても同じである. 例えば,

```
let a = 13;
let b = 3;
document.write(a+b); // 16
```

```
let a = "13";
let b = 3;
document.write(a+b); // 133
```

```
let a = 13;
let b = "3";
document.write(a+b); // 133
```

```
let a = "13";
let b = "3";
document.write(a+b); // 133
```

を比較せよ.

4.9. **数値型と文字列型の変換.** 数と数字は異なると言った. しかし場合によっては, 数字を数として読みたいことや, 逆に, 数を数字として読みたいことがある. つまり, 型の変換が必要となる. javascipt ではそのための**関数**が用意されている.

```
let a = "13";
let b = 3;
document.write(a+String(b)); // 133
document.write("<br>");
document.write(Number(a)+b); // 16
```

`String()` は数値を数字に, `Number()` は数字を数値に変換する関数である. `Number()` は頻繁に用いるが, `String()` の代わりに次のように書くことも多い.

```
let a = 13;
let b = 3;
document.write(a+""+b); // 133
```

問題 1. これは空文字列の巧妙な利用である. ここまでの学習で, なぜこれが連結として機能するか説明せよ.

```
let a = 13;
let b = 3;
document.write(""+a+b); // 133
document.write("<br>");
document.write(a+""+b); // 133
document.write("<br>");
document.write(a+b+""); // 16
document.write("<br>");
document.write(""+(a+b)); // 16
```

5. 入力

ここでは, 入力として,

- prompt による入力

を扱う. これ以外にも様々な入力があるが発展的なのでここでは扱わない.

5.1. **prompt による入力.** 例えば,

```
  let userName = prompt("はじめまして.\n 名前を教えてください.");
  document.write("こんにちは."+userName+"さん！");
```

のような利用が典型的である.

補足 8. ここで利用している\n は**改行コード**である. prompt() と alert() の内部では改行が
では行なえない. その代わりに改行コードを利用する. なお, 逆に document.write() の内部では改行コードが無効なので, やはり
で改行するしかない. このあたりの理屈は本論と関係ない議論をしないといけないので, そういうものだと思うことにしておこう.

また,

```
  let radius = prompt("円の面積を計算します.\n 半径を入力してください.");
  document.write("面積は"+(3.141592*(radius**2))+"です.");
```

は数学教育上の利用価値が高い. ただし, この書き方は「本当は」よくない.

ひとつ目の例からもわかる通り, prompt() で入力した値は文字列型を持つ. つまり, ふたつ目の例においても, 変数 radius の値は文字列型なのである. 実際,

```
  let upperBase = prompt("下底 3, 高さ 4 の台形の面積を計算します.\n 上底を入力してください.");
  document.write("面積は"+((upperBase+3)*4/2)+"です.");
```

では期待通りの結果が得られない. これは, upperBase が文字列だからである. そのため, upperBase+3 では文字列の連結が行なわれてしまったのである.

5.2. **文字列 (数字) に対する演算.** 先に述べた通り, 数字と数の足し算は, 数が数字に変換されて文字列として連結する. しかし, これは足し算のときの例外処理で, それ以外の演算では, 数字が数に変換されて数として演算する:

```
  let a="13";
  let b=3;
  document.write(a+b); // 連結 133
  document.write("<br>");
  document.write(a-b); // 差 10
  document.write("<br>");
  document.write(a*b); // 積 39
  document.write("<br>");
  document.write(a/b); // 商 4.333333333333333
  document.write("<br>");
  document.write(a%b); // 余り 1
  document.write("<br>");
  document.write(a**b); // 冪乗 2197
```

補足 9. 足し算が例外, というより, それ以外が例外だというべきだ.

問題 2. ふたつ目・みっつ目の例はどうすべきだったのだろうか.