コース例: JavaScript によるプログラミング入門

久野 靖

2019.9.7

基本方針

具体的な教材・コース内容の例として「JavaScript によるプログラミング入門」を示す。想定学年は小学校 5・6 年であり、次のことを基本方針としている。

- 変数と手続きによる抽象化を自分のツールとして獲得することが目標。制御構造は変数と組み合わせることで導入。
- 自分の思ったものを作る題材として canvas グラフィクスを採用。
- つまづきを少なくするため Web ブラウザ上で動く実習サイトを使用。

本来であれば (大学生向けなら) 制御構造の次に「配列」を取り扱うが、ここではスキップし、#1: 変数と手続き、#2: 制御構造、#3: グラフィクスの 3 回ぶんの内容として示す。[1]

[1] 実習には「JavaScript 練習ページ」「同 canvas 機能つき」を用いる。同ページは当面の間以下に置く。
https://www.edu.cc.uec.ac.jp/~ka002689/sprosym19/

#1: 変数による値の計算

JavaScript ではプログラムは「関数」と呼ばれるかたまりを組み合わせて作っていきます。関数とは、ひとまとまりの機能を作り出すプログラムに名前をつけたものと考えてください。たとえば、三角形の面積を計算する、という例で考えます。そのための関数は次の形になります。

```
function triarea(w, h) {
  var s = (w * h) / 2;
  return s;
}
```

1行目の「`function triare(w, h)`」で関数の定義を始め、`triarea` という名前をつけています (**triangle** の **area**—面積—のつもりでつけました)。`w` と `h` は関数に与えるパラメタ (計算のつど様々に変化させられる値) です。ここでは三角形の面積なので、`w` は底辺の長さ、`h` は三角形の高さのつもりです。続く「{ ... }」が関数の範囲で、その中には「文 (命令)」がいくつも入れられます。文は並んでいる順番に動作していきます。

#1: 変数による値の計算 (2)

最初の文は「`var s ...`」で、まず「変数」を用意します。変数とは、計算に使う値を覚えておく「いれもの」であり、パラメタも変数の一種として働きます。変数に値を入れる (覚えさせる) ことを「代入」と呼びます。「`s = 計算式;`」により、値を計算して、その結果を変数 `s` に代入しています。[2] このプログラムの場合は計算式は (`w * h`) つまり `w` と `h` を掛けて (*はかけ算を表す)、結果を **2** で割るという内容です。なお、代入は何回でもおこなえ、最後に入れた値が覚えられています。また、変数の名前を書くことで、そこに覚えられている値が取り出されます。[3]

2 番目の文は、`s` に入っている (前の文で計算した) 面積を「結果として返す」働きをします。関数で計算した結果を見るには、**return** で返させる方法と、(後で出てきますが)「`document.writeln(...)`」で途中結果を表示させる方法とが使えます。

[2] JavaScript では「=」は「代入する」という動作を表し「等しい」という条件は「==」のように等号 2 つで表します。
[3] JavaScript では加減乗除算はそれぞれ+、-、*、/で表し、式を 1 行で書くために、適宜 (...) で囲みます。

#1: 変数による値の計算 (3)

ところで、**return** の後に書くものは任意の計算式でよいので、上の例では変数 s の値を取り出して返していましたが、次のプログラムのように面積を計算してそのまま返す (変数 s を使わない) ように書くこともできます。

```
function triarea2(w, h) {
  return (w * h) / 2;
}
```

このように、「正しい」プログラムはいく通りもの方法で作ることができるのが普通です。

#1: 変数による値の計算 (4)

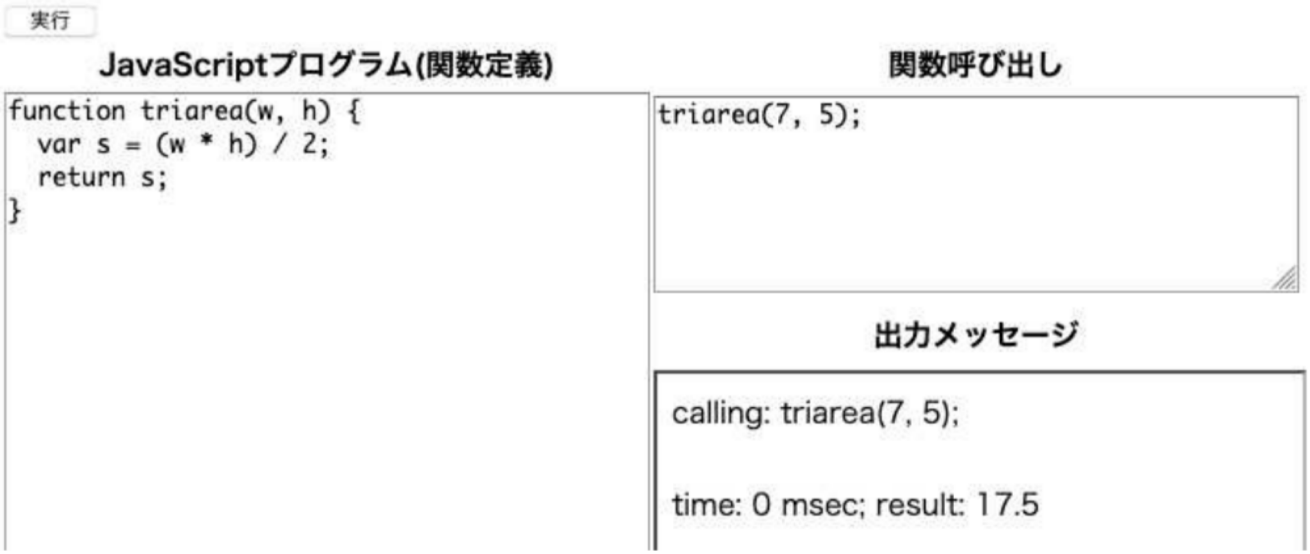


図 1: ブラウザ上の JavaScript 実行環境の例

プログラムを動かすには「練習ページ」を使ってください。練習ページでは、「関数定義」のところに作成した関数を書き込み、「関数呼び出し」のところに「`triarea(7.0, 5.0)`」のようにパラメタとして与える値をつけた関数 (関数呼び出し) を書き込みます。そこで「実行」ボタンを押すと、「出力メッセージ」のところに計算結果が表示されます。

#1: 変数による値の計算 (5)

演習1 練習ページに上で示した「三角形の面積計算」関数の好きな方を打ち込み、動かしてみなさい。プログラムの一部をわざと違うように直した場合どうなるかも観察しなさい。

演習2 次のような関数を作りなさい。呼び出して実行して結果を確認すること。

a. 2つの数を受け取り、和を計算する。(よかったら差、積、商も)。
b. 演算子「%」は剰余 (割った余り) の演算である。剰余を計算する関数を作って結果を試しなさい。
c. 円柱の底面の半径と高さを受け取り、体積を計算する。円周率は3.14であるものとしてよい。
d. 数値xを受け取り、xの8乗 (8回掛けたもの) を計算する。かけ算の回数を少なくできるとよい。
e. 何か自分が面白いと思う計算を行う。

#2: 制御構造

前回のプログラムでは、関数のそれぞれの文は上から順番に1回ずつ実行されていました。しかしプログラムでは、「条件に応じて、ある文を実行したりしなかったりする」「条件が成り立っている間、ある文を繰り返し実行する」などの処理も必要となります。これらの処理のことを (実行の流れを制御することから)「制御構造」と呼びます。

1つ目の制御構造として「枝分かれ」を記述する文であるif文を学びます。

```
if(条件) {
  条件が成り立つ場合の処理
} else {
  上記以外の場合の処理
}
```

#2: 制御構造 (2)

条件としては、a > b (aがbより大きい) などの条件を書くことができます。[4] また、「上記以外の場合」の処理が何もないときは、次のようにelseとそれに伴う処理を書かないでよいです (1行に書いても先の例のように複数の行に分けても構いません)。

```
if(条件) { 条件が成り立つ場合の処理 }
```

[4] 正確には、>(より大)、>=(以上)、<(より小)、<=(以下)、==(等しい)、!=(等しくない) の 6 種類の比較が使えます。

#2: 制御構造 (3)

if文の例として、2つの数値a, bがどれだけ離れているか (たとえばものさしの上で2つの数値を指定したとき、その間がどれだけ空いているか) を計算する関数distを作ってみました。

```
function dist(a, b) {
  var d;
  if(a > b) {
    d = a - b;
  } else {
    d = b - a;
  }
  return d;
}
```

ここでdが「どれだけ離れているか」の度合いです。まず変数dは用意しておき、if文で枝分かれします。もしaとbでaが大きいなら、a-bを計算してdに入れます。それ以外なら、bが大きいかまたはaとbが等しいので、b-aを計算してdに入れます。結果としてはdの値を返せばよいです。

#2: 制御構造 (4)

ところで、おなじ問題は次の書き方にしてもできます。

```
function dist2(a, b) {
  if(a > b) {
    return a - b;
  } else {
    return b - a;
  }
}
```

これは、いちいち変数dに入れる代わりに、計算した結果を直接returnで返します。もう1つ、次のものはどうでしょうか。

#2: 制御構造 (5)

```
function dist3(a, b) {
  var d = a - b;
  if(b > a) { d = b - a; }
  return d;
}
```

これは、とりあえずaの方が大きいとして、dにa-b を入れます。それからif文に入り、もしbの方が大きかったら、そのままでは困るので、b-aを計算してdに「入れ直し」ます。変数には値を何回でも入れられることに注意。ところで、a とbが等しければ? そのときは最初の計算で0が入るでしょうから、そのままでよいわけです。

同じことをする正しいプログラムが何通りにも書けることがお分かりいただけるかと思います。

#2: 制御構造 (6)

演習3 上の例題の好きなものを練習ページで動かし、さまざまな値で動作を確認しなさい。納得したら、次の関数を作りなさい。

a. 2つの異なる数a, bを受け取り、より大きい方を返す。

b. 3つの異なる数a, b, cを受け取り、最大のものを返す。

c. 数xが正なら「'positive'」、負なら「'negative'」、零なら「'zero'」という文字列を返す。[5]

d. 3つの異なる数a, b, cを受け取り、中央の(最大でも最小でもない)ものを返す。

[5] 文字列は文字の並びのことで、両側を'...'または"..."で囲むことで表せます。

#2: 制御構造 (7)

もう1つの制御構造として、「繰り返し」を記述する文であるwhile文を学びます。

```
while(条件) {
  繰り返し実行する処理
}
```

while文では「処理」が何回も実行されるので、少しのプログラムで沢山の計算を行うことができます。その実行のされ方は次のようにイメージするとよいでしょう。

- 「条件」を調べる (成立)。
- 「処理」を実行。
- 「条件」を調べる (成立)。
- 「処理」を実行。
- …
- 「条件」を調べる (不成立)。
- 繰り返しを終わる。

#2: 制御構造 (8)

「終わる」条件を指定したくなりやすいのですが、「終わらない (繰り返し続ける)」条件を指定する、というところがポイントです。

それでは例題として「整数 n を与えたら、1, 2, ... n とカウントする」というのを見てみます。

```
function countup(n) {
  var i = 1;
  while(i <= n) {
    document.writeln(i);
    i = i + 1;
  }
}
```

#2: 制御構造 (9)

document.writeln(...); というのは、プログラム実行の途中で文字や数値を出力します (練習ページだとメッセージ領域に現れます)。最初 i を 1 にしたので、1 が表示されます。次に、i+1 を計算するので、2 が計算されて、それを **i** に代入するので (「=」は代入動作を意味することに注意)、i には 2 が覚えられます。**n** としてある程度大きな値を指定したら、**2** はそれ以下でしょうから、再び document.writeln(...); に進み、2 が表示されます。次の i+1 は 3 が計算され、それを i に覚え直します。このように次々と数値が表示されていきますが、たとえば n として 10 を指定したら、10 を表示したあと、i が 11 になったらそれは「**n** 以下」ではないので、これで繰り返しが終わります。このプログラムは document.writeln(...); で次々に出力するのが仕事なので、**return** で値を返すことはしていません。

#2: 制御構造 (10)

演習4 カウントの例題を動かし確認しなさい。納得したら、次の関数を作りなさい。

a. 1からでなく0からカウントし、nの1つ手前でやめる。

b. 1, 3, 5, ... と奇数だけカウントし、nは表示しない。

c. 1, 2, 4, 8, ... と倍々の値でn未満のものを表示する。

d. n, n-1, ..., 1, 0 と1つずつ減らしながら0までカウントする。

e. 1, 1, 2, 3, 5, ...のように最初の2つは1、以後は「直前の2数の和」をn未満の範囲で表示する。

#3: canvas とグラフィクス

ここまではプログラムの出力は (計算結果を表す) 文字ばかりでしたが、もっと興味を持てる題材として絵の出力 (グラフィクス) を取り上げます。JavaScript を Web ページ上で動かす場合、canvas と呼ばれる四角い出力領域を使ってグラフィクスを出力することが一般的です。このため、練習ページにも canvas がすぐ使えるように「canvas つき練習ページ」を用意しました。このページにはあらかじめ canvas が用意されていて、変数 ctx からその機能が呼び出せます。まず最低限として次の 2 つを使います。

- `ctx.fillStyle = '色指定';` — 塗りつぶしの色を指定。
- `ctx.fillRect(`x`, `y`, `w`, `h`);` — 長方形を指定色で塗る。

色の指定はとりあえず「`'red'`」「`'blue'`」など色の名前を文字列として指定すると思ってください。長方形の指定ですが、x と y は canvas の左側と上側から長方形までの距離、w と h は長方形の幅と高さです。[6]

[6] 大きさの単位は「ピクセル」で、おおよそ 20 ピクセルで 1 センチくらいだと思えばよいでしょう。

#3: canvas とグラフィクス (2)

それでは、複数の正方形を並べて描くという例題を見ましょう。

```
function rect(c, x, y, w, h) {
  ctx.fillStyle=c; ctx.fillRect(x,y,w,h);
}
function picture() {
  var i = 10;
  while(i < 400) {
    rect('blue', i, 10, 20, 20);
    i = i + 30;
  }
}
```

#3: canvas とグラフィクス (3)

まず rect という関数がありますが、これは色 c を使って x, y, w, h の位置/大きさの長方形を塗りつぶします。内容は上で説明した canvas の機能を順に呼び出すだけですが、すぐ次の絵を作る関数を分かりやすく書くために、この関数を作りました。これが打ち込めたところで、「rect('red', 50, 50, 80, 40);」などとして呼び出して動作を確認してみてください (色、位置、大きさを色々変えてみてください)。

次の関数 picture が絵を作り出す本体で、変数 i を 10, 40, 70, 100, ... と増やしながら、横から i、上から 10 の位置に幅・高さともの 20 の青い正方形を描いています。このプログラムを動かした様子は図 2 のようになります。このように、1 つの四角を描くだけなら手で描いた方が速くても、多数の四角を正確に規則的に描くような場合はプログラムを作る方が便利なのです。

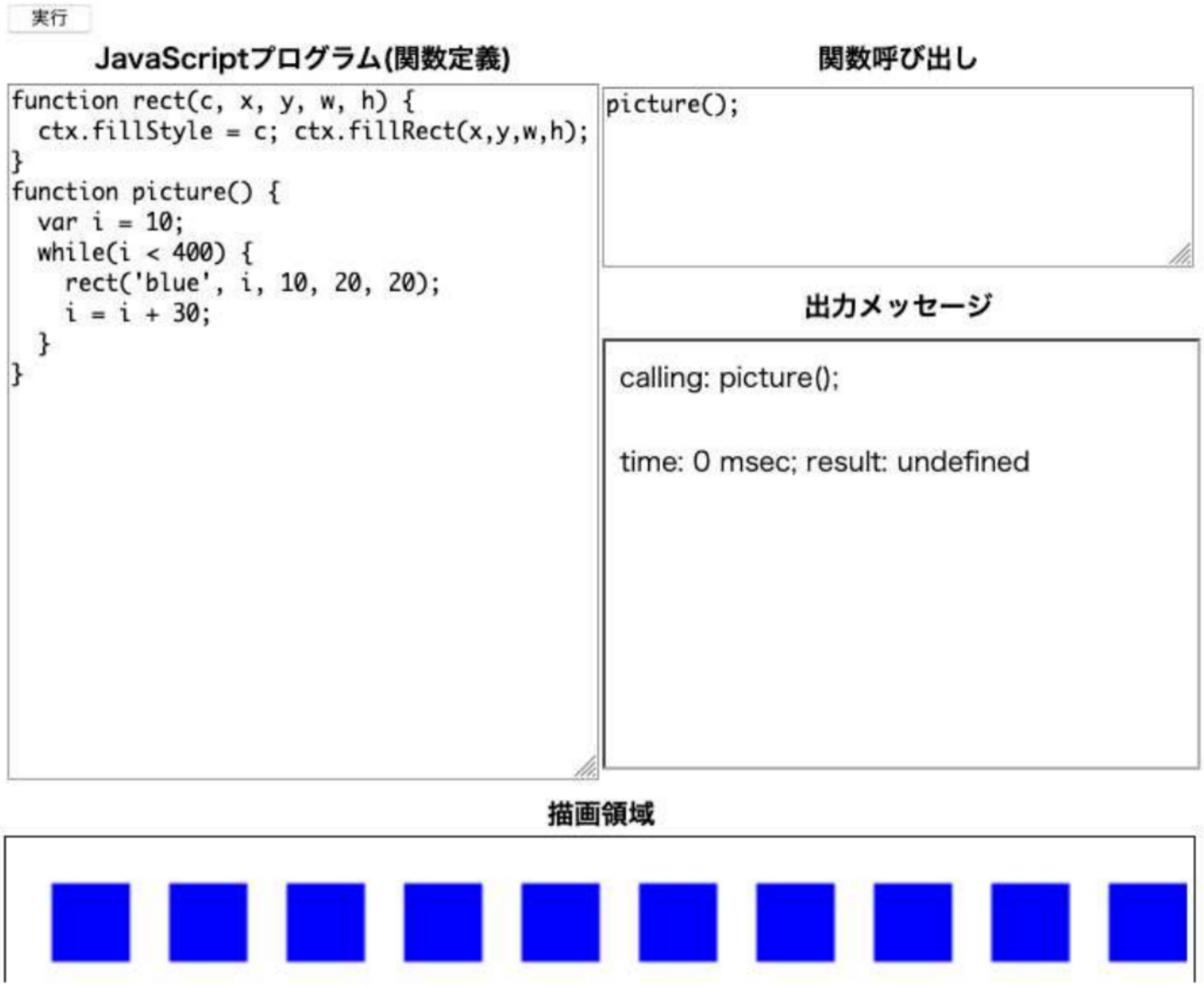
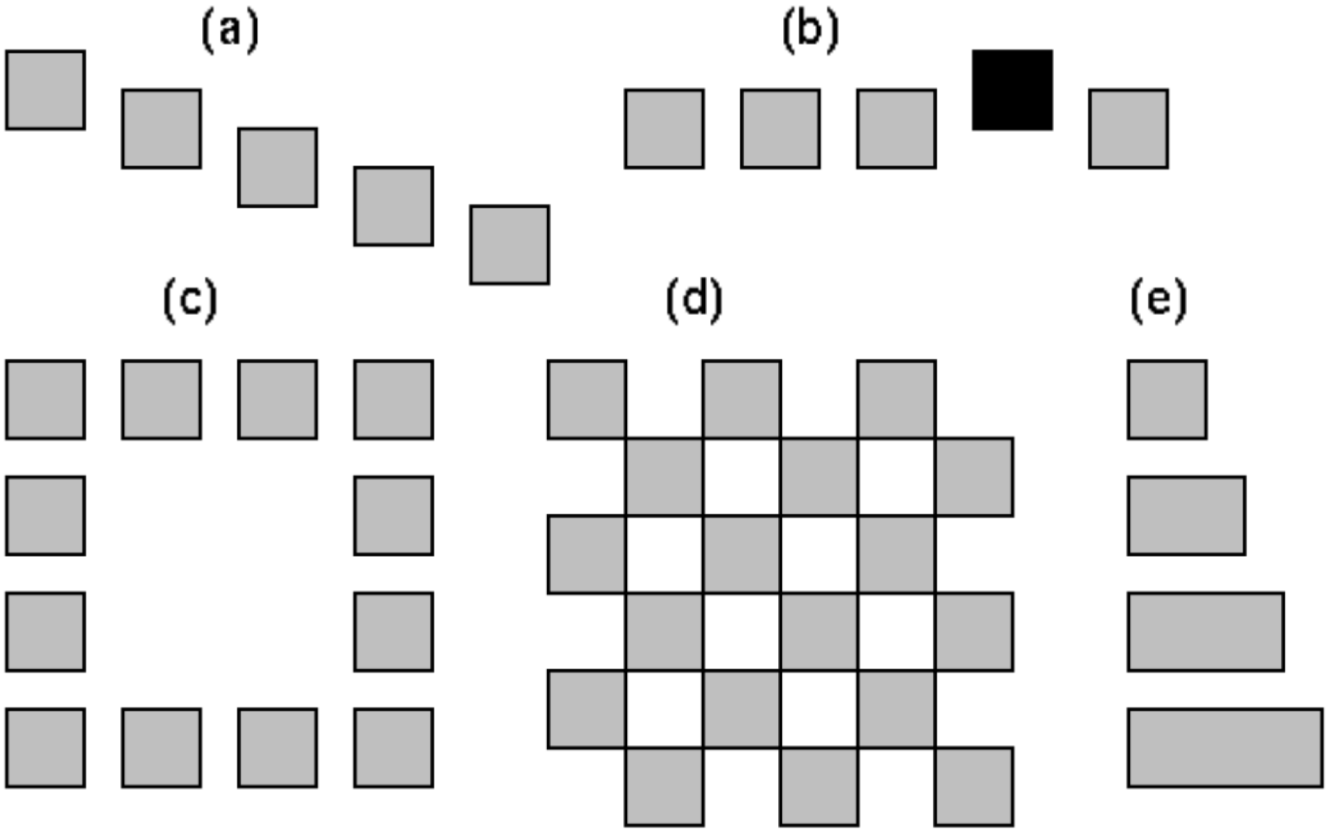


図 2: canvas を併設した実行環境

#3: canvas とグラフィクス (4)

演習 5 「多数の正方形」の例題を動かしてみなさい。動いたら、色や正方形の位置、大きさをさまざまに変えて試してみなさい。納得したら、次の図のような絵を作ってみなさい (個数や配置は好きにアレンジしてよい)。



#3: canvas とグラフィクス (5)

ところで、色が「真っ赤」「真っ青」などではあまりきれいではありません。もっと様々な色を出すには、ちょうど絵の具を混ぜ合わせるように、赤/緑/青の色をさまざまに混ぜ合わせます。そのような指定のため、関数 rgb を作りました。

```
function rgb(r, g, b) {
  r = r%256; g = g%256; b = b%256;
  return 'rgb('+r+','+g+','+b+')';
}
```

この関数は赤/緑/青の強さを **0**〜**255** の範囲の数値で指定しますが、間違って大きな数値を渡してしまった場合でも大丈夫なように **256** で剰余を取ってから使います (つまり **255** を越えると **0** に戻ります)。そのあと、「'rgb(r,g,b)' という文字列を作って返しますが、この文字列は **canvas** の色指定としてそのまま使うことができます。[7]

[7] 「+」演算は数値に対しては足し算をしますが、文字列に対しては「文字列をくっつける」操作になります。

#3: canvas とグラフィクス (6)

このrgbを使って、先の例題の四角を描くところを次のように直してみました。その結果できた絵は図3のようになりました。

```
rect(rgb(i, 100+i, 100), i, 10, 20, 20);
```

図 3: 色を変化させてみる

演習6 色が連続的に変化するような絵を作ってみなさい。

演習7 自分が楽しいと思う絵を描くプログラムを作りなさい。