Chapter 1 (NUPSC 2018 講習資料)

§1.1. プログラムと式

NUPSC では皆さんにたくさんのプログラムを書いて頂きます。 ではプログラムは一体なにでしょうか。 実際のところ、「プログラム」という言葉を厳密的に説明するのは難しいので、 例をいくつかみて、先ずは感覚で覚えておきましょう。

例えばこの文書の下にある青い箱で囲んでいるブロックには一つのプログラムがあります。

```
1 (+ 3 10)
```

```
13
```

このブロックは二つの部分があり、上の部分にある `(+ 3 10)` は正にプログラムにあたります。 では下の部分は何でしょう。 下の部分は実際そのプログラムの実行結果となります。

このプログラムの意味をみてみましょう。 結果からみてすでに気付いているかも知れませんが、 このプログラムの意味は「3と10を'+'(足し併せる)したもの」となります。

このように、実行したら結果の「値」が得られるもの、 あるいは視点を変えて何らかの「値」(このケースでは「3」)を結果として「持つ」プログラムの「文」は、我々通常「式」と呼びます。 これは数式と似たようなものです。

実際数式のように部分式を含んだプログラムの式も書けます。 例えば

```
1 (* 2 (+ 1000 9))
```

```
2018
```

聡明な君ならきっとこの式の意味を理解しているでしょう。 そう、この式は「1000と9を'+'した結果と、2を'*'した結果」を意味しています。

§1.2. Clojure プログラムを実行する

§1.1. では `(+ 3 10)` のような式というものをプログラムとして説明しましたが、 より厳密的にいいますと、このような式は ClojureScript というプログラミング言語の文(つまり ClojureScript で書かれたプログラム)となります。

では ClojureScript のプログラムをどうやったら上のように実行できるでしょうか。

実は我々は既に ClojureScript のプログラムを実行してきました。 上にもあった青い箱で囲んでいるブロックはこれから「コードブロック」と呼びましょう。 又一つのコードブロックの上の部分をソース、下の部分を結果と呼びましょう。

```
(+ 3 10)
```

```
13
```

上に一番最初と同じコードブロックを用意しました。 このコードブロックをすこし変更してみて、「3+20」を計算するようなプログラムに変更してみましょう。 ソースを編集してみると、結果は編集して直ぐ変ったことが分かるはずです。

実はこのウェブページには KLIPSE というツールを介して、 皆さんのブラウザーで ClojureScript を直接実行できるようにしています。

NUPSC の資料のページではないところで ClojureScript を実行したい場合、ClojureScript の ホームページ で書かれている手順に従い ClojureScript をパソコンにインストールするか、 この オンラインの ClojureScript の処理系 を使えばよいでしょう。 残念ながら両方とも日本語の説明がないようです。 これはある程度しようがないことであって、 実際コンピュータサイエンスを本格的に学ぶ上では英語は必須スキルとなっています。

演習1a.

下にあるコードブロックに数式「4*4 + 3*3」を表わすプログラムを書け。

```
(+ (* 4 4)
   (* 3 3))
```

```
25
```

演習1b.

下にあるコードブロックに 黄金比 を計算する数式を書け。 但し割り算は `(/ 4 2)`、平方根は `(Math/sqrt 2)` のように計算できます。

```
(/ (+ 1 (Math/sqrt 5)) 2)
```

```
1.618033988749895
```

§1.3. 関数 (Function)

`Math/sqrt` や `+`、 `*` のようなここまで `(Math/sqrt` *何か*`)` のように使っていたものは ClojureScript では「関数 (function)」と呼ばれるものです。

今まで使ってきました関数は全て ClojureScript にもともとあるものですが、 新に定義することもできます。

例えば平方を取るような関数を新たに定義したいとしましょう。 先ず使いかた(「呼びかた」ともいう)を `(square 4)` のようにしましょう。 つまり `(square 4)` をもって「4の平方」を表わすにるすことです。 このような関数は以下のプログラムで定義できます。

```
(defn square [x]
  (* x x))
```

```
#'cljs.user/square
```

上のコードブロックの結果は `#'cljs.user/square` になっていることに注目しましょう。 先頭の `#'cljs.user/` は一旦無視するとして、 その結果は実際 `square` という名前の付いた関数になります。

このソースは実際二つのことを行っています。 一つ目のことは「xという引数を取ったら `(* x x)` を返すような関数」を作ることで、 二つ目のことは「その関数を `square` に名付けることです。

つまり `defn` というものを使えば新たに関数を定義することができることになります。 実際この定義されていた関数を使ってみましょう。

```
(defn square [x]
  (* x x))
(square 4)
```

```
16
```

上のコードブロックの結果は恐らく予想通り `16` になっているでしょう。 一つ付け加えて説明したいのは、 ClojureScript のプログラムは複数の式であっても構いません。但しその場合全体のプログラムの結果、つまり *値*は最後の式の値になります。

さて、`defn` というものはどう使えばいいかを説明しましょう。 先ずは「defn」の記号の由来は「define function」、 つまり「関数 (function) を定義 (define) する」であることをいっておきましょう。 まさに名前通りの働きですね。

そして `defn` の使いかたは以下のようなものです

`(defn` *新しい関数の名前* `[`*引数1 引数2 引数3* `...]` *引数を使った計算式*`)`

我々の `square` の例では*新しい関数の名前*は「square」で、*引数*は「x」一個だけあって、 *計算式*は `(* x x)` となっています。

演習1.2a.

以下のプログラムを修正し、平面上両点の距離を計算する関数を作りなさい。

```
(defn square [x] (* x x))
(defn distance [x1 y1 x2 y2]
  (Math/sqrt (+ (square (- x1 x2))
                (square (- y1 y2)))))
```

```
#'cljs.user/distance
```

演習1.2b.

上で作った関数を用いて点(1, 7)と点(8, 9)の距離を求めなさい。

```
(defn square [x] (* x x))
(defn distance [x1 y1 x2 y2]
  (Math/sqrt (+ (square (- x1 x2))
                (square (- y1 y2)))))
(distance 1 7 8 9)
```

```
7.280109889280518
```

§1.4. 条件式

もし現時点で絶対値を計算する関数を定義しなさいというお題を出したら、君はどうするでしょうか。

実際厳密的には上に紹介していたClojureScriptの機能を用いて、既に絶対値(absolute value)を求める関数を定義することができます。けれどそれは歪な方法であって、きれいな数学の定義のように絶対値関数は上に紹介されている機能だけでは書けません。新しい機能を利用して、絶対値を求める関数は以下のように定義できます。

```
(defn abs [x]
  (if (≥ x 0)
    x
    (- x)))
```

```
#'cljs.user/abs
```

ここでは`if`というものと、`>=`というものを使用しています。`if`は「条件文」、あるいは「if文」と呼ばれるもので、`(if` *条件式* *真節* *偽節*`)`のように三つの引数と取ります。名前の通り、条件文は*条件式*の真偽によって結果が変わる式であって、もし*条件式*の結果は「真(true)」であれば条件文全体の結果は*真節*の結果となり、逆にもし*条件式*の結果は「偽(false)」であれば条件文全体の結果は*偽節*の結果となります。

そして`>=`というものは「述語」と呼ばれるものの一つであって、渡された第一引数

は第二引数以上であれば「真」を返し、そうでなければ「偽」を返すようなものです。

ClojureScript、そして一般的に「真」と「偽」のいずれという値は「真偽値」と呼ばれ、 コンピュータサイエンスでは「boolean値」、あるいは「bool値」、「ブール値」ともしばしば呼びます。 「述語」というのは何らかの引数を受け取り、真偽値を返すようなものを指す専門用語であって、 コンピュータサイエンスではよく見かけます。

ClojureScript では >= 以外にも >、<=、<、=のような様々な述語があります。 又述語といっても関数の一種なので、自作することもできます。

演習1.3.

以下のプログラムを修正し、 二次方程式 $ax^2 + bx + c = 0$を解く関数を定義せよ。 但し実数根が存在しない場合0を返し、複数の実数根が存在する場合大きい方の根のみを返すものとする。

```
(defn delta [a b c]
  (- (* b b)
     (* 4 a c)))
(defn qsolve [a b c]
  (if (< (delta a b c) 0)
    0
    (/ (+ (- b) (Math/sqrt (delta a b c)))
       (* 2 a))))
```

```
#'cljs.user/qsolve
```