

第 **10** 日

1時限目
tic_tac_toeの解説

Enumerableモジュールのメソッド

ヒント

自作クラスであっても、eachメソッドを実装して列挙可能にすれば、Enumerableモジュールをミキシンできます。

サンプルプログラムのapp¥models¥tic_tac_toe.rbでは、複数のEnumerableモジュールのメソッドを利用しています。「Enumerable」を辞書で引くと、「可算の」などとなっていますが、ここでは「列挙可能なオブジェクト」といった意味で使われています。

これらのメソッドは、Rubyの特徴であるブロック引数を取ることや、コンパクトに記述できることから、多くのプログラムでよく使われています。ですから、これらのメソッドについて知っておくことは、Rubyのプログラムを記述する上でも、ほかのRubyプログラムを読む上でも、重要です。

また、Enumerableモジュールは、Arrayクラス、Stringクラス、Rangeクラス、Hashクラス、Dirクラスなど、eachメソッドを持つクラスにミキシンされているため、クラスを横断して利用できます。その意味でもこれらのメソッドの価値は大きいと言えます。

findメソッド

要素に対してブロックを適用し、真となった最初の要素を返します。

```
def get_empty_cell(a) —— 最初のnilのセルを返すメソッド
  a.find do |e|
    get(e).nil? —— 引数にしてgetメソッドを呼び出した
                  結果がnilとなる最初の要素を返す
  end
end
```

POINT ワンポイント・アドバイス

上記のリストのように、ブロック内の式が1つの場合、do~endの代わりに{|}が利用される傾向があります。その場合、メソッド定義内に、「a.find {|e| get(e).nil?}」が入ります。

injectメソッド

引数で指定した初期値をブロックの結果で置き換えながら、全要素に対してブロックを適用した結果を返します。

```
def count_player(line, player) —— lineで示した座標の配列内のplayerで
                                  指定したプレイヤーの数を返す
  line.inject(0) do |r, pos| ——
    ブロックの引数のrの初期値はinjectメソッドの引数の0になる。
    以降、ブロックを実行する都度、その結果が次のブロック実行時の
    rの値に設定される。posは列挙される要素
    if get(pos) == player —— もし該当座標のプレイヤーがplayerと等
                              しなければ、それぞれ以下の値を返す
      r + 1 —— 1を加えた値を返す
    else
      r —— そうでなければ、現在の値を返す
    end
  end —— injectメソッドの戻り値は最終的なrの値
end
```

POINT ワンポイント・アドバイス

Arrayクラスのオブジェクトの==メソッドは、異なるオブジェクトであっても要素が等しければ真となります。

```
【例】 a = []; b = []; a << 1; b << 1; a == b ——
aとbは別々に作ったArrayクラスのオブジェクトだがどちらも要素が1なので真
```

この処理をeachメソッドで書き換えた例を示します。

```
def count_player(line, player)
  r = 0 ————— あらかじめ変数の初期化 (=宣言) が必要
  line.each do |pos|
    if get(pos) == player
      r += 1 ————— 代入演算が必要
    end
  end
  r ————— 戻り値の指定が必要
end
```

find_allメソッド

要素に対してブロックを適用し、真となったすべての要素の配列を返します。

```
points = line.find_all { |e| get(e).nil? } └──
セルがnilの全座標の配列をpoints変数に設定する
```

このメソッドをeachで書き換えると次のようになります。

```
points = [] ————— あらかじめ変数の初期化 (=宣言) が必要
line.each do |e|
  if get(e).nil?
    points << e ————— 追加操作が必要
  end
end
```

include?メソッド

引数で指定した値と等しい要素が存在すれば真を返します。

```
if BORDERS.include? point ————— BORDERS配列の中にpointと  
同じ座標があれば真
```

このメソッドをeachで書き換えると次のようになります。

```
found = false ————— あらかじめ変数の初期化 (=宣言) が必要
BORDERS.each do |e|
  if point == e
    found = true
    break
  end
end
if found
  :省略
```

以上見てきたように、これらのメソッドを利用することで、eachメソッドでは必須となる変数の初期化 (=宣言) や破壊的な操作 (代入や配列への追加など) の記述が不要となります。

このことは、変数名の書き間違いによる誤代入や、初期化 (宣言) 忘れによる変数への設定内容の消失、初出変数によるエラーを避けることができることを意味します。このことは、書き間違いによるつまらないバグを減らし、プログラムの品質の向上に貢献します。

また、eachメソッドと異なり、メソッドの戻り値に意味があるため、直接ifの条件式として記述したり、右辺値として扱うこともできます。

特にinjectメソッドや、ここでは使われていないcollect (map) メソッドは、一見すると何を行っているかわかりにくいため、Rubyプログラミングに慣れないうちは敬遠されがちです (eachメソッドで代替できるから、という理由もあるかもしれません)。しかし、これらのメソッドを利用することでプログラムの品質を上げることができるのですから、避けるべきではありません。実際のところ、Enumerableモジュールのメソッドを多用するRubyらしいプログラミングというのは、Rubyを使ったバグの入りにくいプログラミング記述の結果です。