テキスト処理'15 ＃2 – テキスト処理とプログラム構造

久野　靖 *

2015.6.2

1　はじめに

前回はファイルの読み書きと行単位の選択までで終わってしまいましたが、今回はその積み残しとしてパターンの話してから各行の中のデータを抽出して処理することを行います。関連して、平均、分散 (標準偏差)、中央値、四分位数 (ヒンジ) など基本的なパラメタの計算も (アルゴリズムを書く練習として) 扱います。

あと、前回は説明しませんでしたが、Ruby のドキュメント (マニュアル) はこちらから見られます。自分が使っているバージョンのものを見てください (この授業の範囲では違いはほとんどないはずです)。

```
https://www.ruby-lang.org/ja/documentation/
```

このほか、この講義ではデータの抽出関係しかやらないので、ある程度体系的に本で勉強したい人のために参考書を 2 件紹介します。別に無理に買う必要はないと思います。

- Chris Pine, 長尾訳, 初めてのプログラミング第 2 版, オライリー, 2010.
- Yugui, 初めての Ruby, オライリー, 2008.

前者は Ruby を用いたプログラミングの入門書で、丁寧に 1 歩ずつ書かれています。自習に向いていると思います (もちろん、読み飛ばしただけではプログラミングの学習は無理ですから、それなりに時間は掛ける必要がありますが)。後者は Ruby の特徴的な部分を中心に丁寧に解説されていて、プログラミングの初歩は終わった人向けです。なお、「オブジェクト指向 (クラス)」のところは本講義では使わないので飛ばしてもよいです。

2　パターンマッチと正規表現

前節では単に「何行目か」だけに基づいて行の並び替えや連結をしていましたが、当然ながら実際のデータ処理では「行の中身 (つまり文字列の内容)」も見ながら処理をするのが普通です。

では文字列の内容をどのようにして調べるのでしょうか。古典的なプログラミング言語だと、文字列に対してできることは「長さを調べる」「i 文字目を取り出す」「$i \sim j$ 文字目を取り出す」「文字 (列) が同じかどうかを調べる/(辞書順の) 大小を調べる」くらいで、これらを組み合わせて必要な処理をするのは結構大変でした。

これに対し、最近の言語では**正規表現** (regular expression) と呼ばれる形の**パターン**が使えるので、文字列の内容を調べるのがとても簡単になっています。具体的には、入力データがパターンにマッチするかどうかを判定できますし、マッチした時はついでにその必要な部分を抽出して以後の処理に使う、という形でデータの取り出しにも活用できます。Ruby では正規表現は「`/.../`」で囲んで表します (他の多くの言語でもそうです)。

*経営システム科学専攻

```
if /パターン/ =~ 文字列      ←「=~」はマッチ演算子
   あてはまった場合の処理
else
   あてはまらなかった場合の処理
end
```

正規表現については計算機科学基礎でもやりましたが、簡単におさらいしておきましょう。正規表現 e は次のいずれかの形になります。

* c --- 通常の (特別な意味を持たない) 文字はその文字そのものにマッチ
* . --- 任意の 1 文字にマッチ
* [$c_1c_2c_3\ldots$] --- c_1, c_2, c_3 ... のいずれかにマッチ
* [c_1-c_2] --- コードが c_1 から c_2 の範囲の文字にマッチ
* [^...] --- ... で指定した文字群以外の文字にマッチ (文字クラス)
* e* --- e を 0 回以上繰り返したものにマッチ
* e? --- e を 0 回または 1 回繰り返したものにマッチ
* e+ --- e を 1 回以上繰り返したものにマッチ
* e_1|e_2 --- e_1 または e_2 のいずれかにマッチ
* ^e --- 行頭の e にマッチ
* $e$$ --- 行末の e にマッチ
* (e) --- e にマッチし、なおかつマッチしたものを覚える

また、メタ文字列 (\+1 文字) も使用できます。

* \n --- 改行文字
* \r --- 復帰文字
* \s --- 空白文字全般 (文字クラス)
* \d --- 数字 (文字クラス)
* \w --- 英数字 (文字クラス)

なお、「[]|/()」などの特別な意味を持つ文字を普通の文字として使う場合は \ を前置します。\ を指定したい場合は「\\」です。

3 行内容のマッチに基づく処理

ではいよいよ、パターンマッチを使った処理の例を見ましょう。「継続行」機能を持つようなファイル形式があります。たとえば、行末に「\」があるとその行は次の行とつながっている (「\」と続く改行が無かったことになる)、というのが代表例です。たとえば次のようになります。

```
aaa bbb \          =>  aaa bbb ccc
ccc                    ddd eee fg
ddd ee\
e fg
```

このようなファイルを読み込み、継続行をくっつけて長い行にする処理を作成してみます。

```
def readdata
  r = []
  open('test2.txt') do |f|
    f.each_line do |s| r.push(s.chop) end
  end
  return r
end
def writedata(a)
  a.each do |s| puts(s) end
end
def contlines(a)
  r = []; l = ""
  a.each do |s|
    if s =~ /\\$/ then
      l += s.chop
    else
      l += s; r.push(l); l = ""
    end
  end
  if l != "" then r.push(l) end
  return r
end

$data1 = readdata
$data2 = contlines($data1)
writedata($data2)
```

contlines の中身ですが、変数 l は「現在作成中の行」に対応し、最初は空文字列です。渡された配列の各文字列について、「行末が\かどうか」を正規表現で判定しています。「\」だった場合は、その文字を削除した上で変数 l の後ろにくっつけます。そうでなかった場合は、そのまま l の後ろにくっつけ、l を r の末尾に追加してから空文字列にリセットします。なぜこれでいいか、よく考えてください。

ループが終わった後の if 文は何でしょうか。ループが終わって出て来たときに、l が空文字列でないのは、最後の行が「\」で終わっていた、ということです。この場合の処理はどうすべきか (次の行に続くはずなのにその次の行が無いので) 困るのですが、内容が失われるのも困りそうなので、残った l の内容を最後に追加しています。[1]

演習 4 上の例題をそのまま打ち込み、動かしてみて、動作を確認しなさい。

演習 5 次のようなデータの加工をおこなうメソッドを書きなさい (紙に書いて検討すること。実際に打ち込んで動かすのは後にする)。

a. 行の先頭が「#」になっている行をコメント行として削除する。★

b. 行の先頭が空白になっている行は前の行の続きとして前の行の後にくっつける。★[2]

c. a. のようにコメント行を削除しつつ、例題のように行末の「\」を次の行への継続として扱う。「\」で終わる行の後ろにコメント行があった場合、コメント行は無視し、次のコメントでない行をくっつけること。

d. a. のようにコメント行を削除しつつ、b. のように空白で始まる行を前の行の継続として扱う。コメント行の後に空白で始まる行があった場合はその行は通常の行として扱う (これにより、空白で始まる行が出力できるようになる)。

e. 数値の並びから成るファイルがあるので、その合計を求める。ただし a. のようなコメント行も含まれているので、それは無視する。

[1] ところで、このコードでは最後がこのような「\」で終わる行なのに無くなってしまう場合というのもあるのですが (ある意味ではバグ)、どういう場合か分かりますか。

[2] なお、文字列 s の 2 文字目以降を取り出すのは「s[1..-1]」でできます。

f. 数値の並びから成るファイルがあるので、その合計を求める。ただし a. のようなコメント行があったら、そこでそこまでの小計を出力し、小計は 0 とする。最後に残った小計と合計とを出力する。

演習 6 演習 5 で作成したプログラムを実際に動かせ。ただし、動かす前にテストデータと「そのテストデータを与えた場合どのような結果になるか (想定出力)」を用意し、動かした後想定出力との一致を確認すること。

4 データの抽出

今度は、どのような内容を扱うかを見ていただくため、例題の説明をしましょう。2012 年の各国穀物生産高の次のようなデータがファイル `grain2012.txt` に入っているものとします。

```
# 順位  国名          2012 年            注
   1    中国          540,830,000
   2    アメリカ      356,961,850
   3    インド        286,500,000
   4    ブラジル       89,908,244
   ...
 182    グアム                 46
 183    ジブチ                 14
```

これはサイト `http://www.globalnote.jp/post-1279.html` から表をコピーペーストして持って来たものです。取り方によっては空白でなくタブ区切りで取れると思いますが、それでも別に構いません。ここから、データ (国名と生産高) を取り出て扱いたいとします。[3]

このためには、前回やった正規表現を使って各行とのパターンマッチを行った上で、「その一部を取り出す」ことを行います。前回扱った正規表現の説明の中に、「(…)」で囲むと内容を覚える、というのがありましたが、何のことか分からなかったかと思います。実は今回のような用途にはこれが役立つのです。

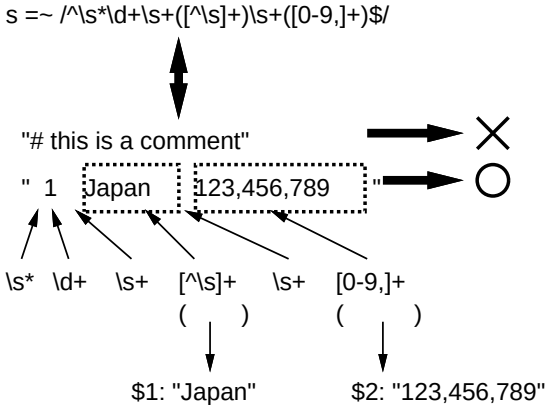

図 1: 正規表現によるマッチと抽出

図 1 に今回の例題で扱う正規表現を用いた例を示します。正規表現は次のものです。

```
\s*\d+\s+[^\s]+\s+[0-9,]+
```

[3]本当は Web ページなのでスクレイピングするといいのですが、まだそんな難しいところまでやっていないので、とりあえずコピペでテキストを取りました。

これは左から、「空各が 0 個以上」「数字が 1 個以上」「空白が 1 個以上」「空白でない文字が 1 個以上※ 1」「空白が 1 個以上」「0〜9 と, が 1 個以上※ 2」という意味になります。まず、行全体 (/^〜$/) がこのパターンにあてはまるか否かを調べ、否の場合はコメントかエラー行として処理します。次にあてはまる場合ですが、これらの部分のうち※ 1 と※ 2 の部分が扱いたいデータ (国名と数量) なので、そこを抽出したいわけです。それには、抽出したい部分を「;(...);」いくつでもで囲んでおくと、マッチが成功したときに左から順に特別な変数$1、$2、…に対応する値が入っています。

では、これを使ってまずは「データを抽出して表示する」例題プログラムを示します。

```
def readdata
  r = []
  open('grain2012.txt') do |f|
    f.each_line do |s| r.push(s.chop) end
  end
  return r
end
def pickdata(a)
  r = []
  a.each do |s|
    if s =~ /^#/ then
      # do nothing
    elsif s =~ /^\s*\d+\s+([^\s]+)\s+([0-9,]+)/ then
      name = $1; quant = $2
      r.push([quant.gsub(/,/, '').to_i, name])
    else
      $stderr.puts("wrong format: #{s}")
    end
  end
  return r
end
def showdata(a)
  a.each do |d| puts "#{d[0]} #{d[1]}" end
end

$data1 = readdata
$data2 = pickdata($data1)
showdata($data2)
```

まず readdata は前回と同じで、全部の行を配列に読み込みます。次の ppickdata がデータを取り出すところなので、これを説明します。

- 変数 r に空の配列を入れる。これにデータを追加していって最後に返すのは前回同様。
- 次に、入力の各行について順番に処理。
- まず、先頭が#の行はコメントとして何もしない (無視)。
- そうでなくて、指定パターンに合う場合はデータを抽出。この正規表現は、「(…)」を無視して考えると、次のように読めます。「行の先頭から、空白文字が 0 個以上、数字が 1 個以上、空白文字が 1 個以上、非空白文字が 1 個以上、空白が 1 個以上、そして数字またはカンマが 1 個以上並んでいる。」先のデータを見るとそのようになっていることが分かると思います (\s が意味する空白文字にはタブ文字も含まれます)。
- 次に「(…)」ですが、「非空白が 1 個以上」のところと、「数字またはカンマが 1 個以上」のところが囲まれています。ということは、パターンがあてはまったときには、変数$1 に「非空白が 1 個以上」の部分、変数$2 に「数字またはカンマが 1 個以上」の部分が格納されています。そこで、あてはまり OK の場合は、この 2 つを変数 name、quant に取り出し (すぐ取り出さないと$1 等は次のパターンマッチで別の値になってしまうので)、続いてこれらの値を「2 要素の配列として」r に追加します。つまり r は配列の配列になります。

- なお、名前はそのまま文字列でいいのですが、穀物生産量は数値なので整数にしたいです。そのため、まず.gsub(/,/, '') で全部のカンマを空文字列にして (つまり削除して)、それから .to_i で文字列を整数に変換します。カンマを削除しておかないと数値への変換が正しく行われないので注意。
- コメントでもデータでもない場合…というのは、何かがおかしいわけですが、それを無視したらデータ処理の失敗に気付き損ねます。そこで、いずれでもない場合は「wrong format:」というメッセージに続いてその行を表示します。この表示は標準エラー出力に対して行います (出力データと混ざらないため)。
- 最後に return でできあがった結果を返します。

最後の表示をおこなう showdata は非常に簡単で、配列の各要素 (というのは数量と国の名前の 2 要素から成る配列) を順に取り出し、それを 1 つの文字列として (中に配列の 0 番目と 1 番目の要素を埋め込んで) 出力しています。2 つのデータは空白で区切られていますが、「,」にすれば CSV 形式になるわけです。

ついでに余談 (というか以下の演習のヒント) ですが、ではこのデータの穀物生産量の合計を求めるにはどうするでしょうか。解答を示しておきます。

```
def totaldata(a)
  sum = 0.0
  a.each do |d| sum += d[0] end
  return sum
end
```

演習 7 例題を打ち込み、データファイルをコピーしてそのまま動かしてみなさい。次に、例題のパターンの「\s+」となっているところをただの「\s」にして動かしてみなさい。一部の行で抽出に失敗するはずです。なぜ失敗するのか検討してみなさい。

演習 8 次のような値を計算するメソッドを作成しなさい。データは大きいものから順に並んでいるものとしてよいです。いずれも紙に書いて作成すること。

a. 平均値を求める avgdata。★
b. 標準偏差を求める sddata。平方根は Math.sqrt(…) で計算できます。★[4]
c. 最大値と最小値を求める maxmindata(2 つの値を並べた配列として返す)。★
d. 中央値を求める mediandata。[5]
e. 第 1 四分位数、中央値、第 3 四分位数を求める quantdata(3 値を並べた配列を返す)。[6]

演習 9 演習 8 で作成したコードを実際に打ち込んで動かし、テストしなさい。どのようなテストデータを与えたらよいか検討し、その想定出力も用意すること。

5 整列メソッド.sort の使用方法

先のデータは既に生産高の大きい順に並んでいましたが、並んでいないデータが渡される場合もあります。また、小さい順に並べたい、国名の順に並べたい、などの場合も考えられます。その場合は配列のメソッド.sort をうまく使うことで対応できるので、これについて説明しておきましょう。

まず、配列 a が数値から成っていて、数値の昇順 (小さい順) に並べる場合は次の 2 つのいずれかを使えば良いです。

[4] 平均を m とすると分散は $\sigma_i(x_i - m)^2/n$ となり、標準偏差は分散の平方根です。
[5] 中央値は大きさ順に並べた中央の値であり、データ数が奇数ならその中央のデータの値、偶数なら中央の 2 つの値の平均値となる。
[6] 簡単のため、四分位数としてヒンジ (中央値より大きい値の中央値、中央値より小さい値の中央値) を求めるのでよい。

```
a.sort!      # これまでの配列を書き換え
b = a.sort   # 新しい配列を別に作る
```

以下では書き換え型だけで説明しますが、「!」が無い版は元のデータは変更せずに新しい配列が返される以外、同じです。データについては先の例題の「数量, 国名」の 2 要素の配列が並んだ配列であるものとします。まず、数量の昇順に並べる場合は次のようになります。

```
$data2.sort! do |x, y| x[0] <=> y[0] end
```

x と y には 2 つのデータ (いずれも並べ替える要素、つまりここでは 2 要素の配列) が渡され、それに対して次の値を返す必要があります (正確には 1 や −1 でなくて正や負の数値でもよい)。

```
1  ... x の方が値が大きい (並び順が後)
0  ... x と y の値は同じ (並び順で一緒)
-1 ... x の方が値が小さい (並び順が先)
```

ここで「<=>」は 2 つの数値や文字列を (大小順や辞書順で) 比較する演算子で、まさに上の値を返すので、これで昇順に並ぶことになります。降順 (大きい順、元のデータの状態) にしたければ次の通り。

```
$data2.sort! do |x, y| y[0] <=> x[0] end
```

また、数量でなく国名の順にしたければ次のようにします。

```
$data2.sort! do |x, y| x[1] <=> y[1] end
```

6　ヒストグラムと累積度数 (割合)

今度は同じデータを用いて、もう少し違う処理をしてみます。具体的には、各国のデータをヒストグラムにしてみました。

```
 (same for readdata, pickdata)
def histdata(a, w, m)
  r = Array.new(m+1, 0)
  a.each do |d|
    n = d[0].to_i / w
    if n >= r.size then n = r.size-1 end
    r[n] += 1
  end
  return r
end
def showhist(h, w)
  h.each_index do |i|
    printf("%12d %4d\n", w*i, h[i])
  end
end

$data1 = readdata
$data2 = pickdata($data1)
$hist1 = histdata($data2, 5000000, 20)
showhist($hist1, 5000000)
```

readdata と pickdata は同じなので略。histdata は、データの配列に加え、いくつの幅ごとにまとめるかを表すパラメタ w、度数を何件とるかを表すパラメタ m を取ります (最後のより大きいものは全部最後の度数に加えます)。内容は次の通り。

- 要素数 m+1 で初期値 0 の配列を用意。

- 各データごとにデータを w で切捨て割り算した値を n とし (ただし最大を超える場合は最大にする)、配列の n 番目を 1 増やす。
- 最後に度数の配列を返す。

メソッド showdata は各データ行を整形して出力するものです。printf については C 言語のものと同じ機能ですね (「%nd」は整数を n 文字ぶんの幅で右寄せして表示)。

演習 10 例題を打ち込んでそのまま動かしなさい。また、w や m を変更してわかりやすい値はどのくらいか検討しなさい。他のデータをコピーしてきて同様にやってみなさい。

演習 11 各国のデータに次の値を計算して追加し、表示できるようにしなさい (プログラムは紙で書くこと。変更/追加のないメソッドはわざわざ書かなくてもよい)。

a. 生産量上位の国から順に累積生産量を計算し、並べて表示する。
b. 生産量上位の国から順に累積生産量率 (全合計で 100%になる) を計算し、並べて表示する。
c. b. と同じだが、上位 10 国で表示を打ち切る。

演習 12 演習 10 で作成したものを実際に動かしてみなさい。

7 パネルデータの処理

今度はもう少し実務に近そうなデータとして、パネルデータの抜粋を処理してみます (元が非常に大きなものなので、演習用に特定の 1 日ぶんだけ取り出して小さくしてあります)。形は次のようになっています。パネル番号とはポイントカードの ID のような、個人を識別する番号です。

パネル番号, 店番号, レシート番号, 発行時, 品数, 金額, 商品名

各欄の区切りは「,」で、商品名は文字列ですが、それ以外の欄はすべて整数になっています。1 文字目が「#」の行はコメントです。冒頭の数行を見てみましょう。

```
#panel,shop,receipt,time,count,amount,itemname
15360,20,643,20,1,138,ほうれん草
22568,22,7093,11,1,98,長ねぎ
28192,22,5043,16,1,98,長ねぎ
```

商品名の中に「,」は含まれないので、単純に「,」の箇所で区切ることでデータを抽出できます。では、このデータの中から各購入について品数、金額、商品名を取り出し、そのまま打ち出すという例をやってみましょう。

```
def readdata
  r = []
  open('ITEM1.csv') do |f|
    f.each_line do |s| r.push(s.chop) end
  end
  return r
end
def pickdata(x)
  r = []
  x.each do |s|
    if s =~ /^#/ then
      # do nothing
    elsif s =~/^(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(.+)$/ then
      a,b,c,d,e,f,g = $1,$2,$3,$4,$5,$6,$7
      r.push([a.to_i, b.to_i, c.to_i, d.to_i, e.to_i, f.to_i, g])
    else
      $stderr.puts("wrong format: #{s}");
    end
  end
  return r
end
def show1(d)
  d.each do |a|
    printf("%3d %8d %s\n", a[4], a[5], a[6]);
  end
end

$data1 = readdata
$data2 = pickdata($data1)
show1($data2)
```

上のコードを動かしたところの冒頭部分を示します。

```
1      138 ほうれん草
1       98 長ねぎ
1       98 長ねぎ
1       98 長ねぎ
1       78 長ねぎ
1       68 長ねぎ
2       52 たまねぎ（バラ用）
3       96 たまねぎ（バラ用）
...
```

演習 13 このデータに対して次の処理をするプログラムを紙に書きなさい。readdata、pickdata は同じままでよいはず。

a. この日の売り上げ合計を表示。★
b. この日に売れた全商品の平均販売価格を表示。★
c. この日に売れた最も高額な商品の販売価格価格と商品名を表示。
d. この日に売れた最も安い商品の販売価格価格を商品名を表示。
e. この日の販売で同じ商品のまとめ買い数の最大と商品名を表示。

演習 14 紙に書いたプログラムを実際に動かして動作を確認しなさい。

8 ハッシュを使った集計

次は「一番多く買った人は誰か」を調べてみます。パネルデータは品目ごとにバラバラになっていますから、それぞれの人 (パネル番号) ごとに累計していき、累計が一番高い人を見つける必要があ

ります。この「○○ごとに累計」のような処理をおこなうのに適した機能が「ハッシュ(hash)」と呼ばれるデータ構造です。ハッシュは配列によく似ていますが、配列が添字が順に並んだ整数に限られるのに対し、バラバラの値や文字列など任意の値が添字にできます。概念としては、図 2 のような鍵 (key) と値 (value) の対になった「表 (テーブル)」を想像すればいいでしょう。

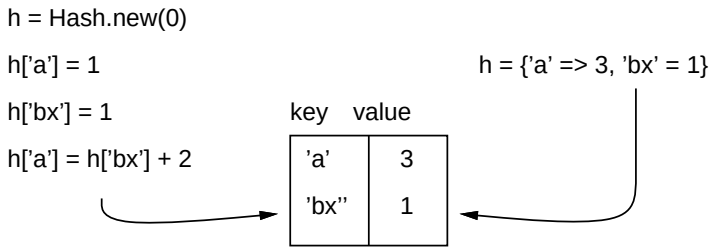

図 2: ハッシュの概念

ハッシュを作るのには `Hash.new(0)` のようにして空っぽのハッシュを作ることができます。その後、図 2 左では文字列を添字として値を代入したり、現在の値を取り出したりしています。「どこの値を」取り出すかが添字 (鍵) によって指定されるわけです)。ところで、`Hash.new(0)` の 0 は何かというと、まだ値を入れていない添字を指定して取り出した場合に出て来る値を 0 にする、という指定です (今回の例では金額などを扱うので 0 が出て来ると便利)。なお、ハッシュの別の作り方として図 2 右のように「{ 鍵 => 値, 鍵 => 値, … } 」のように複数の鍵と値を指定して一度に作るという方法もあります。

では、ハッシュを使って先の「最も多く買った人とその金額」をやってみます。

```
def readdata
  r = []
  open('ITEM1.csv') do |f|
    f.each_line do |s| r.push(s.chop) end
  end
  return r
end
def pickdata(x)
  r = []
  x.each do |s|
    if s =~ /^#/ then
      # do nothing
    elsif s =~/^(\d+),(\d+),(\d+),(\d+),(\d+),(\d+),(.+)$/ then
      a,b,c,d,e,f,g = $1,$2,$3,$4,$5,$6,$7
      r.push([a.to_i, b.to_i, c.to_i, d.to_i, e.to_i, f.to_i, g])
    else
      $stderr.puts("wrong format: #{s}");
    end
  end
  return r
end
def show2(d)
  total = Hash.new(0); max = 0; maxid = 0
  d.each do |a|
    total[a[0]] += a[5]
    if total[a[0]] > max then max = total[a[0]]; maxid = a[0] end
  end
  printf("max = %d, panel = %d\n", total[maxid], maxid);
end

$data1 = readdata
$data2 = pickdata($data1)
show2($data2)
```

説明は次の通り。

- 最初にハッシュを作り total という変数に保持。また現在の最大値と最大値を達成したパネル id を保持する変数を用意 (最初は 0)。
- 各データについて「数量×金額」を計算し、現在のその人の total に加算する。
- 加算した結果をこれまでの最大値を比較し、より大きければ最大値を更新するとともにそのパネル id も記録
- 処理が終わったら最後に金額とパネル id を出力。

演習 15 同じデータに対してハッシュを使用して次の処理をするプログラムを紙に書きなさい。readdata、pickdata は同じままでよいはず。

a. 最も売れた件数が多い商品と売れた件数を求めなさい。★
b. 上記に加え、売れた合計金額も求めなさい。★
c. この日に買物した人数を求める (ヒント: ハッシュに対して「`.size`」を参照するとテーブルの項目数が分かる)。
d. 購入金額が最も多い時間帯を求める。
e. 一人でもっとも多くの品物を購入した人のパネル ID と何品購入したかを求める (ヒント: ハッシュの各値としてさらにハッシュを入れる)。

演習 16 紙に書いたプログラムを実際に動かして動作を確認しなさい。

9 第 2 回レポート課題

2 つ以上の演習のプログラムを選んで「プログラムを実際に動かす」課題をおこない、レポートとして提出しなさい。レポートではプログラムごとに次の内容を含むこと。

- 学籍番号、氏名、提出日付 (これらは最初に 1 回でよい)
- 問題の再掲
- 作成したプログラムとその説明
- 実行結果と考察

とくに考察において、「ロジックのどこが難しいか、どのように工夫したか」をきちんと書くこと。レポート課題は次回授業開始時までに、`gssm.k-kiso` に投稿すること。ただし★がついている問題 (易しい問題) を含む場合はは本日中。