「学習者コーパス論」講義ノート

―Rによるテキスト処理入門―

１．Rによる簡単な計算

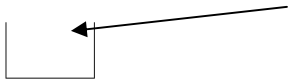
（注）#はコメント

```
+    # 足す
-    # 引く
*    # かける
/    # 割る
^   # べき乗
sqrt(x)    #ルート (1/2乗)
log(x)    #xの自然対数を取る。底はe
log2(x)   #xの対数を取る。底は2
```

●計算例

```
> 1 + 2
[1] 3
> 4 -3
[1] 1
> 5 * 6
[1] 30
> 8 / 2
[1] 4
> 2^3
[1] 8
> sqrt(9)
[1] 3
> log(2)
[1] 0.6931472
> log2(8)
[1] 3
```

２．変数の作成と値の代入



変数という入れ物に値を代入する。

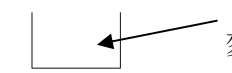
#変数名は、英数字と_.が使える。R のコマンドと重複しないように。

```
> abc <- 3    # abc という変数を作り、3 を代入。
> abc    # abc の中身を表示させる。
[1] 3    # 3 が入っている。
> abc <- 300   # abc の値に 300 が上書きされる。
> abc    # abc の中身を表示。
[1] 300   #中身が 300 に変わった。

> efj <- 6
> jke <- 2987
> ls()   # これまでに作った変数一覧を表示。
[1] "abc" "efj" "jke"

> rm(jke)   #変数 jke を削除。
> ls()
[1] "abc" "efj"
```

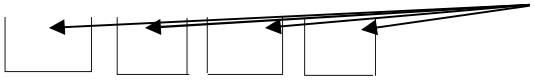
- 文字と数字を区別する

```
> namae <- "sugiura"   # namae には sugiura という文字列が入る。
>class(abc)    #class(変数名)で、変数に入っているものが数字なら numeric を、
              # 文字なら character を返す。
[1] "numeric"

> class(namae)
[1] "character"
```

- 変数（配列）の作成と値の代入



変数の場合



配列の場合

```
> kazu <- c(2,4,6,8)
```

```
                    # kazu という配列を作り、1 つ目の要素に 2、2 つ目の要素に 4
                    # 3 つ目に 6, 4 つ目に 8 を代入。
> kazu  # kazu の中身を表示
[1] 2 4 6 8

> names <- c("I","you","he")   # 配列の中身は文字列でもいい。
> names
[1] "I" "you" "he"
> length(names)   #length(配列名)で、該当配列の要素数を返す。
[1] 3   # 配列 names の要素は、 "I" "you" "he"の 3 つ。
```

３．作業スペース・履歴の保存と読み込み

#　作業を中断する場合、作業スペースと履歴を保存しておくと、作成した変数や使用したコマンドが保存されるので便利。

- 作業スペースの保存
  1．R のメニューバーより、「ファイル→　作業スペースの保存」を選択
  2．保存したい場所を指定し、ファイル名を付ける。この時、拡張子を.RData にする。

- 履歴の保存
  1．R のメニューバーより、「ファイル→　履歴の保存」を選択
  2．保存したい場所を指定し、ファイル名を付ける。拡張子を.Rhistory にする。

- 作業スペースの読み込み
  1．R のメニューバーより、「ファイル→　作業スペースの読み込み」を選択
  2．読み込みたいファイル名を選択し、「開く」をクリック
  3．もしくは、読み込みたいファイルを保存したフォルダを開き、該当ファイルをダブルクリック。

- 履歴の読み込み
  1．R のメニューバーより、「ファイル→　履歴の読み込み」を選択
  2．読み込みたいファイル名を選択し、「開く」をクリック

４．文書ファイルの読み込み

- 方法 1

```
scan (file ="ファイルの場所と名前", what="char")
```

#ファイルをフルパス付きで指定
what="char"で、データ形が文字列であることを指定

(実行例)
```
> scan(file= "E:/NS001.txt", what="char")
```

- 方法2

```
scan(choose.files(), what="char")
```

#GUIウィンドが現われるので、ファイルを選択

(出力例)
```
Read 586 items
  [1] "@Begin"              "@Participants:"      "NS001"
  [4] "@Age:"               "46"                  "@Sex:"
  [7] "M"                   "@L1:"                "AmE"
 [10] "@FatherL1:"          "AmE"                 "@MotherL1:"
 [13] "AmE"                 "@AcademicBakground:"  "MA"
```
（以下省略）

- セパレータの変更

```
> scan(choose.files(), what="char", sep="¥n")
```
#セパレータを改行マーク(¥n)に変更。
#デフォルトではスペース。

(出力例)
```
 [1] "@Begin"
 [2] "@Participants:¥tNS001"
 [3] "@Age:¥t46"
 [4] "@Sex:¥tM"
 [5] "@L1:¥tAmE"
 [6] "@FatherL1:¥tAmE"
 [7] "@MotherL1:¥tAmE
```
（以下省略）

５．データの切り出し

- 読み込んだファイルをRのデータとして保存

```
> ns.lines <- scan(choose.files(), what="char", sep="¥n")
```
#ns.linesは変数(配列)の名前。好きな名前を付けてよい。

(出力例)
```
Read 81 items
```
#ns.linesを作成し、81の要素を読み込んだことを報告。

● 欲しい部分だけ取り出す

head(*変数名, 要素数*)　#変数の先頭部分を指定した要素数表示させる。

tail(*変数名, 要素数*)　#変数の末尾部分を指定した要素数表示させる。

```
> head(ns.lines)  #ns.lines の先頭部分を表示させる。デフォルトは 6 つの要素。
```

(出力例)

```
[1] "@Begin"                "@Participants:¥tNS001" "@Age:¥t46"
[4] "@Sex:¥tM"              "@L1:¥tAmE"             "@FatherL1:¥tAmE"
```

```
> head(ns.lines,14)  #ns.lines の先頭 14 行を表示させる。
```

(出力例)

```
 [1] "@Begin"
 [2] "@Participants:¥tNS001"
 [3] "@Age:¥t46"
 [4] "@Sex:¥tM"
 [5] "@L1:¥tAmE"
 [6] "@FatherL1:¥tAmE"
 [7] "@MotherL1:¥tAmE"
 [8] "@AcademicBakground:¥tMA"
 [9] "@OtherLanguage:¥tnone==;none=="
[10] "@Topic:¥tdeath penalty"
[11] "@WritingProficiency:¥t"
[12] "@Comments:¥tNoTitle"
[13] "@Coder:¥t2006-10-28 DataInputBy MURAO Remi;"
[14] "@Version:¥t1.1"
```

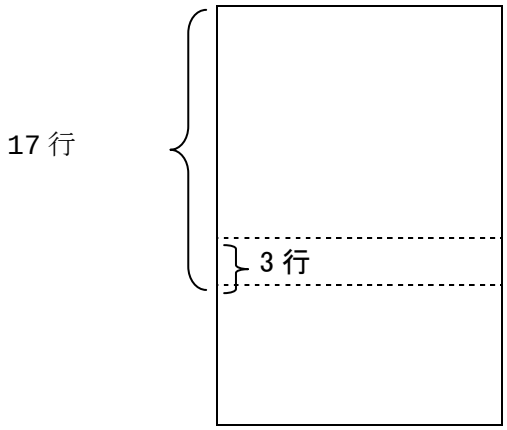
```
> tail(ns.lines,67)
  #ns.lines の後ろから 67 行を表示させる (ヘッダーを取り除く)。
```

(出力例)

```
 [1] "*NS001:¥tWhy I Support The Death Penalty"
 [2] "%COM:¥t"
 [3] "%par:"
 [4] "*NS001:¥tI support the death penalty."
 [5] "%COM:¥t"
 [6] "*NS001:¥tHowever, I think it should only be given to murderers
or those who physically torture other people to such an extent that they
```

are permanently crippled."
（以下省略）

> tail(head(ns.lines,17),3)　　＃15-17 番目の要素だけを取り出したい場合。
先頭から 17 番目までを取り出し、その最後から 3 つの要素を取り出し。
#基本的に、コマンドは組み合わせることができる。



(出力例)
[1] "*NS001:¥tWhy I Support The Death Penalty"
[2] "%COM:¥t"
[3] "%par:"

● 配列要素の取り出し
配列名[*要素番号*]

> ns.lines[79]　#79 番目の要素のみ取り出し。

(出力例)
[1] "*NS001:¥tHe should have been executed."

> ns.lines[c(77,78,79)]　#77, 78, 79 番目の要素のみ取り出し。

(出力例)
[1] "*NS001:¥tTo say this sentence was too light would be an understatement."
[2] "%COM:¥t"

```
[3] "*NS001:¥tHe should have been executed."
```

```
> ns.lines[77:79]  #77 から 79 番目の要素のみ取り出し。
```

(出力例)

```
[1]  "*NS001:¥tTo  say  this  sentence  was  too  light  would  be  an
understatement."
[2] "%COM:¥t"
[3] "*NS001:¥tHe should have been executed."
```

- grep で、該当文字列を含む要素のみ取り出し

grep (“*文字列*”, *変数名*)　#要素の番号のみを返す。

```
> grep("NS001", ns.lines)
```

#変数 ns.lines について、”NS001”を含む要素番号を表示。

(出力例)

```
[1]  2 15 18 20 22 24 26 28 30 33 35 37 39 42 44 46 48 51 53 56 58 60
63 65
[25] 67 69 71 73 75 77 79
```

```
> ns.lines[grep("NS001", ns.lines)]
```

#配列 ns.lines から、上記番号の要素のみ取り出し。

(出力例)

```
 [1] "@Participants:¥tNS001"
 [2] "*NS001:¥tWhy I Support The Death Penalty"
 [3] "*NS001:¥tI support the death penalty."
```

（以下省略）

```
> grep("NS001", ns.lines, value=T)
```

#value=T で、要素番号ではなくて中味を表示。

#ns.lines[grep("NS001", ns.lines)]と同じ。

```
> tail(grep("NS001", ns.lines, value=T),-1)
```

上で取り出した 1 番目の要素は、作文の本文ではないため、tail(*配列名*,-1)で、1 番目削除。

(出力例)

```
 [1] "*NS001:¥tWhy I Support The Death Penalty"
 [2] "*NS001:¥tI support the death penalty."
 [3] "*NS001:¥tHowever, I think it should only be given to murderers or those who physically torture other people to such an extent that they are permanently crippled."
```

（以下省略）

```
> ns.data <- tail(grep("NS001", ns.lines, value=T),-1)
```

#欲しい部分だけ取り出すことができたので、新しい変数名 ns.data を付けて保存。

６．単語リストを作る

- ns.data 中身の確認

```
> ns.data
```

(出力例)

```
 [1] "*NS001:¥tWhy I Support The Death Penalty"
 [2] "*NS001:¥tI support the death penalty."
 [3] "*NS001:¥tHowever, I think it should only be given to murderers or those who physically torture other people to such an extent that they are permanently crippled."
```

（以下省略）

- 単語をバラバラにする

strsplit (*変数名*, *“セパレータ”*)

(実行例)

```
> strsplit (ns.data, " ")  #ns.data の要素をスペースで分割。
                           #要素ごとに処理される。つまり、list になっている。
```

(出力例)

```
[[1]]
[1] "*NS001:¥tWhy" "I"          "Support"     "The"
[5] "Death"       "Penalty"
[[2]]
[1] "*NS001:¥tI" "support"    "the"       "death"     "penalty."
```

```
[[3]]
 [1] "*NS001:¥tHowever," "I"                "think"
 [4] "it"              "should"           "only"
 [7] "be"              "given"            "to"
[10] "murderers"        "or"               "those"
[13] "who"             "physically"        "torture"
[16] "other"           "people"            "to"
```

（以下省略）

● リストの要素をバラバラにする

`unlist (`*変数名*`)`

(実行例)

```
> ns.data.list <- strsplit (ns.data, " ")
```

#まずは上での処理結果を ns.data.list という変数に保存。

```
> unlist (ns.data.list)
```

ns.data.list のリストの要素をバラバラにする。

unlist(strsplit(ns.data, " "))でも同じ結果。

(出力例)

```
  [1] "*NS001:¥tWhy"      "I"                "Support"
  [4] "The"              "Death"            "Penalty"
  [7] "*NS001:¥tI"        "support"           "the"
 [10] "death"            "penalty."          "*NS001:¥tHowever,"
 [13] "I"                "think"            "it"
```

（以下省略）

● ばらばらにした単語をソートする

`sort (`*変数名*`)`

(実行例)

```
> sort(unlist(ns.data.list))
```

(出力例)

```
  [1] "¥"eye-for-an-eye¥"" "*NS001:¥tA"        "*NS001:¥tAfter"
  [4] "*NS001:¥tAnd"      "*NS001:¥tAs"       "*NS001:¥tBy"
```

（中略）
[40] "a"　　"a"　　"accidentally"
[43] "act"　　"after"　　"after"
（以下省略）

● 同じ単語を 1 つにまとめる

unique (*変数名*)

(実行例)
```
> unique (sort(unlist(ns.data.list)))
```
#ソートした単語をタイプ（異なり語）にまとめる。
sort(unique(unlist(ns.data.list)))でも結果は同じ。

(出力例)
[1] "¥"eye-for-an-eye¥"" "*NS001:¥tA"　　"*NS001:¥tAfter"
[4] "*NS001:¥tAnd"　　"*NS001:¥tAs"　　"*NS001:¥tBy"
（中略）
[25] "act"　　"after"　　"again"
[28] "again."　　"all,"　　"all."
[31] "already"　　"also"　　"an"
（以下省略）

● 置換

gsub(元の文字列, 新しい文字列, 対象データ)

#上記単語リストには、*NS001 などの記号類が含まれる。
#これらのいらない文字列を置換により削除する。

（実行例 1）（失敗）
```
> gsub("*NS001:¥t" , "", ns.data)  # "* NS001:¥t"が不要なので、""に置換。
```
→ * NS001:¥t を削除することになる。
¥t はタブを表す正規表現。

（出力例）
以下にエラー gsub("*NS001:¥t", "", ns.data) :
'*NS001:　　' の正規表現が不正です
追加情報: Warning message:
In gsub("*NS001:¥t", "", ns.data) :
regcomp のエラー: ' 先行する正規表現は無効です

エラーの原因は、*NS001 の*が特殊文字だったため。
¥¥を特殊文字の前に付けて、エスケープする。

（実行例 2）（成功）

```
> gsub("¥¥*NS001:¥t" , "", ns.data)  # ¥¥を*の前に付けて、エスケープ。
```

（出力例）

```
 [1] "Why I Support The Death Penalty"
 [2] "I support the death penalty."
 [3] "However, I think it should only be given to murderers or those
who physically torture other people to such an extent that they are
permanently crippled."
```

（以下省略）

```
ns.data2 <- gsub("¥¥*NS001:¥t" , "", ns.data)
```

#記号類を削除することができたので、新しい変数名 ns.data2 で保存。

```
> ns.data2  #中身を確認。
 [1] "Why I Support The Death Penalty"
 [2] "I support the death penalty."
 [3] "However, I think it should only be given to murderers or those
who physically torture other people to such an extent that they are
permanently crippled."
```

（以下省略）

- いらない文字列を削除して、再度単語リストを作る

#以上をまとめると、以下の一行で単語リストができる。

```
unique (sort(unlist( strsplit (gsub("¥¥*NS001:¥t" , "", ns.data), "
"))))
```

(出力例)

```
  [1] "¥"eye-for-an-eye¥"" "24"              "a"
  [4] "A"               "accidentally"     "act"
  [7] "after"           "After"           "again"
 [10] "again."          "all,"            "all."
```

（以下省略）

７．頻度表を作る

table (*変数名*)　# sort をしなくても単語の頻度表を作成。

（実行例)

```
> table(unlist(strsplit (ns.data2, " ")))
```

#単語をバラバラにした後で table を使用。

（出力例）

```
"eye-for-an-eye"            24             a             A
               1             1             9             1
   accidentally           act         after         After
               1             1             2             1
          again        again.           all,          all.
               1             2             1             1
```

（以下省略）

【復習：JPN001.txt の単語頻度表を作る】

```
> scan(choose.files(), what="char", sep="¥n")
```

#まずは、ファイルの読み込み。

(出力例)

```
Read 121 items
  [1] "@Begin"
  [2] "@Participants:¥tJPN001"
  [3] "@Age:¥t20"
```

（中略）

```
 [23] "*JPN001:¥tDoes death penalty really need?"
 [24] "%NTV:¥tIs the death penalty really necessary?"
 [25] "%COM:¥t"
```

（以下省略）

```
> jpn.lines <- scan(choose.files(), what="char", sep="¥n")
```

変数 jpn.lines に代入。

```
> grep("JPN001", jpn.lines) #”JPN001”を含む要素番号を表示。
```

(出力例)
```
[1] 2  23  27  30  33  36  39  43  46  49  52  55  59  62  65  68  71  74
78
[20]  81  84  87  90  93  97 100 103 106 109 112 115 118
```

```
> grep("JPN001", jpn.lines, value=T)
```
#value=T で、要素番号ではなくて中味を表示。
#jpn.lines[grep("JPN001", jpn.lines)]と同じ。

(出力例)
```
 [1] "@Participants:¥tJPN001"
 [2] "*JPN001:¥tDoes death penalty really need?"
 [3] "*JPN001:¥tDoes death penalty really need?"
```
（以下省略）

```
> grep("¥¥*JPN001", jpn.lines, value=T)
```
#@Participants:¥tJPN001 は不要なため、より細かく指定。
#*は特殊文字のため、¥¥でエスケープ。

(出力例)
```
 [1] "*JPN001:¥tDoes death penalty really need?"
 [2] "*JPN001:¥tDoes death penalty really need?"
 [3] "*JPN001:¥tI often hear that death penalty is reasonable for people
whose family member or friend is killed by someone."
```
（以下省略）

```
> jpn.data <- grep("¥¥*JPN001", jpn.lines, value=T)
```
#学習者のデータ行のみを取り出したので、一旦保存。

```
> gsub("¥¥*JPN001:¥t", "", jpn.data)
```
#"*JPN001:¥t"を""に置換（つまり、"*JPN001:¥t"を削除)。

(出力例)
```
 [1] "Does death penalty really need?"
 [2] "Does death penalty really need?"
 [3] "I often hear that death penalty is reasonable for people whose
family member or friend is killed by someone."
```
（以下省略）

```
> jpn.body <- gsub("¥¥*JPN001:¥t", "", jpn.data)
      #学習者の本文のみを取り出したので、一旦保存。
> strsplit(jpn.body, " ")  #jpn.body の要素をスペースで分割。
```

(出力例)

```
[[1]]
[1] "Does"   "death"   "penalty" "really"  "need?"
[[2]]
[1] "Does"   "death"   "penalty" "really"  "need?"
```

（以下省略）

```
> unlist(strsplit(jpn.body, " "))  #リストの要素をバラバラにする。
```

(出力例)

```
  [1] "Does"       "death"      "penalty"    "really"     "need?"
  [6] "Does"       "death"      "penalty"    "really"     "need?"
```

（以下省略）

```
> jpn.words <- unlist(strsplit(jpn.body, " "))
    #上記の結果を jpn.words に一旦保存。

> length(jpn.words)  #jpn.words の要素数を返す（つまり、総語数(token)）。
```

(出力例)

```
[1] 436
```

```
> unique(jpn.words) #同じ単語を１つにまとめる。
```

(出力例)

```
  [1] "Does"       "death"      "penalty"    "really"     "need?"
  [6] "I"          "often"      "hear"       "that"       "is"
```

（以下省略）

```
> length(unique(jpn.words))  #異語数(type)を求める。
```

(出力例)

```
[1] 205
> table(jpn.words) #頻度を数える。
```

(出力例)

```
jpn.words
      "no".        "yes".             a         abuse      accepted
adult
         1            1            6            1            1            1
    against        alive      allowed       almost         also
always
         3            1            1            1            4            1
```

（以下省略）

```
> jpn.table <- table(jpn.words)  #jpn.table に一旦保存。

> write.table(jpn.table, file="jpnwordlist.txt")
    #上記の結果をテーブルの形式でファイル jpnwordlist.txt に保存。
```

jpnwordlist.txt の中味

```
--------------------------
"jpn.words" "Freq"
"¥"no¥"." "¥"no¥"." 1
"¥"yes¥"." "¥"yes¥"." 1
"a" "a" 6
"abuse" "abuse" 1
"accepted" "accepted" 1
"adult" "adult" 1
"against" "against" 3
"alive" "alive" 1
(以下省略)
----------------------
> write.table(jpn.table, file="jpnwordlist2.txt", sep="¥t")
    #セパレータをタブに。
```

jpnwordlist2.txt の中味

```
------------------------
"jpn.words"   "Freq"
"¥"no¥"."     "¥"no¥"."     1
"¥"yes¥"."    "¥"yes¥"."    1
"a"    "a"    6
"abuse"       "abuse"       1
```

```
"accepted"    "accepted"    1
"adult"       "adult"       1
(以下省略)
----------------------
```

```
>    write.table(jpn.table,    file="jpnwordlist3.txt",    sep="¥t",
row.names=F) #行名を無しに。
```

jpnwordlist3.txtの中味

```
-------------------------
"jpn.words"  "Freq"
"¥"no¥"."    1
"¥"yes¥"."   1
"a"    6
"abuse"      1
(以下省略)
----------------------
```

```
>    write.table(jpn.table,    file="jpnwordlist4.txt",    sep="¥t",
row.names=F, col.names=F)  #行・列の名前を無しに。
```

jpnwordlist4.txtの中味

```
---------------------
"¥"no¥"."    1
"¥"yes¥"."   1
"a"    6
"abuse"      1
"accepted"   1
"adult"      1
"against"    3
(以下省略)
----------------------
```

- 大文字・小文字、記号類の処理

#これで単語の頻度表ができたが、問題が2点。

1．大文字・小文字の処理

2．記号類の処理

1. 大文字・小文字の処理

`tolower(`*変数名*`)`　#全て小文字に揃える。

(実行例)

```
> tolower(jpn.words)
```

(出力例)

```
  [1] "does"       "death"      "penalty"    "really"     "need?"
  [6] "does"       "death"      "penalty"    "really"     "need?"
```

（以下省略）

`toupper(`*変数名*`)`　#全て大文字に揃える。

(実行例)

```
> toupper(jpn.words)
```

(出力例)

```
 [1] "DOES"       "DEATH"      "PENALTY"    "REALLY"     "NEED?"
  [6] "DOES"       "DEATH"      "PENALTY"    "REALLY"     "NEED?"
```

（以下省略）

2.　記号類の処理

#ヒント：gsub を使う。例えば、gsub("¥¥.", "", 変数名)で、コンマを削除。

#しかし、記号は色々ある・・・(.,!?"' etc.)。

`gsub("¥¥W", "", `*変数名*`)`　#記号類を全て消す！

#¥w　：ワード構成文字(アルファベット・数字など)

#¥W　：ワード構成文字以外

（実行例）

```
> gsub("¥¥W", "", jpn.words)
```

(出力例)

```
  [1] "Does"       "death"      "penalty"    "really"     "need"
  [6] "Does"       "death"      "penalty"    "really"     "need"
```

（以下省略）

８．R で関数を作成し、TTR, Guiraud Index, 平均単語長、平均文長を出す

- データの準備

```
# NICE から、学習者または母語話者の作文のみを取り出す関数は以下。
# これを R Console に貼り付ける。

myData <- function(){
       lines.tmp <- scan(choose.files(), what="char", sep="¥n")
         #ファイルを選択。
       data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
         #*で始まる行のみ。
       body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
         #行頭のタグを削除。
       body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
}

> jpn.body <- myData()  #画面に入力すると、ファイルを選択する窓が現われる。
    #JPN001.txt を選択し、myData で本文のみを取り出した結果を jpn.body に代入。

(出力例)
Read 121 items

> jpn.body  #中身を確認。
 [1] "Does death penalty really need?"
 [2] "Does death penalty really need?"
 [3] "I often hear that death penalty is reasonable for people whose
family member or friend is killed by someone."
（以下省略）
```

- TTR

```
# TTR = Type/ Token
# 関数 myTTR を実行する前に、以下の動作を確認しよう。

> (jpn.body.lower <- tolower(jpn.body))
  #大文字を小文字に。外の()は、処理と同時に表示。
```

(出力例)
```
 [1] "does death penalty really need?"
 [2] "does death penalty really need?"
 [3] "i often hear that death penalty is reasonable for people whose
family member or friend is killed by someone."
```
（以下省略）

```
> (jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W")))
```
　# 単語をバラバラに。

(出力例)
```
  [1] "does"      "death"     "penalty"   "really"    "need"
  [6] "does"      "death"     "penalty"   "really"    "need"
```
（以下省略）

```
> length(jpn.words) #jpn.words の要素数を求める＝総語数(Token)を求める。
```

(出力例)
```
[1] 463
```

```
> length(unique(jpn.words)) #異語数(Type)を求める。
```

(出力例)
```
[1] 181
```

```
> length(unique(jpn.words))/length(jpn.words) #TTR を出す。
```

(出力例)
```
[1] 0.3909287
```

#以上をまとめた関数が以下。これを R Console に貼り付ける。
```
myTTR <- function(a){
       jpn.body.lower <- tolower(a)
       jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
       length(unique(jpn.words))/length(jpn.words)
}
```

```
> myTTR(jpn.body) #上記関数の a の部分に、jpn.body が入る。
```

(出力例)
[1] 0.3909287

- Guiraud Index

```
# Guiraud Index = Type/ √Token
# Guiraud Index を出す関数は以下。

myGI <- function(a){
       jpn.body.lower <- tolower(a)
       jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
       length(unique(jpn.words))/sqrt(length(jpn.words))
}

> myGI(jpn.body)  #Guiraud Index を出す。
```

(出力例)
[1] 8.411783

- 平均単語長

```
# 平均単語長(AWL) = 総文字数／総語数
# 関数 myAWL を実行する前に、以下の動作を確認しよう。

> (jpn.words <- unlist(strsplit(jpn.body, "¥¥W"))) # 単語をバラバラに。
```

(出力例)

```
  [1] "Does"      "death"     "penalty"   "really"    "need"
  [6] "Does"      "death"     "penalty"   "really"    "need"
```

（以下省略）

```
> length(jpn.words) #jpn.words の要素数を求める＝総語数(Token)を求める。
```

(出力例)
[1] 463

```
> paste(jpn.words, collapse="")
    #jpn.words の要素を全てくっつける。仕切りはなし。
```

(出力例)
[1]
"DoesdeathpenaltyreallyneedDoesdeathpenaltyreallyneedIoftenhearthat
deathpenaltyisreasonableforpeoplewhosefamilymemberorfriendiskilledb
ysomeoneOntheotherhandIals
（以下省略）

```
> nchar(paste(jpn.words, collapse=""))  #文字数を数える。
```

(出力例)
[1] 1908

#以上をまとめた関数が以下。

```
 myAWL <- function(a){
        jpn.words <- unlist(strsplit(a, "¥¥W"))
        nchar(paste(jpn.words, collapse=""))/length(jpn.words)
 }
```

```
> myAWL(jpn.body) #jpn.body の平均単語長を出す。
```

(出力例)
[1] 4.12095

- 平均文長

```
# 平均単語長(ASL) = 総語数／総文数　（それぞれの文が持つ単語数の平均値）
# 関数 myASL を実行する前に、以下の動作を確認しよう。
```

```
> (x <- strsplit(jpn.body, "¥¥W"))
   #jpn.body をワード構成文字以外で分割し、x に代入。
```

(出力例)
[[1]]
[1] "Does"　"death"　"penalty" "really"　"need"

[[2]]
[1] "Does"　"death"　"penalty" "really"　"need"

（以下省略）

```
> x[[1]]  #例えば、x[[1]]の中身は以下。
[1] "Does"   "death"  "penalty" "really"  "need"

> length(x[[1]]) #x[[1]]の要素数（つまり単語数）は以下。
[1] 5

#以上より、それぞれのセンテンスの単語数を出す関数が以下。

for (i in 1:length(x)) {
   # iを1からlength(x)(つまりセンテンスの数)まで1ずつ増やす)。
      print(length(x[[i]]))  #それぞれのセンテンスの単語数をプリント。
}

(出力例)
[1] 5
[1] 5
[1] 19
（以下省略）

# 最後にプリントアウトせず、変数yに入れる関数が以下。
y <- 0  # とりあえず、yを定義。
for (i in 1:length(x)) {
      y[i] <- length(x[[i]])
}

> y  #中身を確認
 [1]  5  5 19 31 15 15 24 22 23 26 21 16 17  7 10 23 12 17 12  9 19 21
23  5  7
[26] 10  4  6 14 13 12
# mean(y)でyの平均値を求めれば、ASLが出る！

#以上をまとめて、ASLを出す関数が以下。
myASL <- function(a) {
      x <- strsplit(a, "¥¥W")
      y <- 0
      for (i in 1:length(x)) {
```

```
        y[i] <- length(x[[i]])
        }
        mean(y)
}

> myASL(jpn.body)
```

(出力例)
[1] 14.93548

- 小島版平均文長（for ループを使用しなくてもできる）

```
myASL2 <- function(a){
        jpn.sent <- strsplit(a, "¥¥W")
        jpn.words <- unlist(jpn.sent)
        length(jpn.words)/length(jpn.sent)
}

> myASL2(jpn.body)
```

(出力例)
[1] 14.93548

- 使用頻度の高い語を調べる

```
# 使用頻度の高い上位 10 単語を出す関数
myTop10 <- function(a){
        jpn.body.lower <- tolower(a)   #小文字に揃える。
        jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
          #単語をバラバラに。
        jpn.words <- jpn.words[jpn.words != ""]  #空の要素を削除。
        head(sort(table(jpn.words), decreasing = T), 10)
          #頻度の高い順から top 10 を表示。
}

> myTop10(jpn.body)
```

(出力例)

```
jpn.words
   is   the   that    to    it    in people   kill    and  death
   29    17    14    14    13    10    10     8     7     7
```

```
# 下の関数では、使用頻度の高い単語を何位まで出すか指定できる。
myTop <- function(a,b){   #処理したい変数をaに、何位まで出すかをbで指定。
       jpn.body.lower <- tolower(a)
       jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
       jpn.words <- jpn.words[jpn.words != ""]
       head(sort(table(jpn.words), decreasing = T), b)
}
```

```
> myTop(jpn.body,20)   # jpn.bodyを対象に、高頻度語20位を出す。
```

(出力例)

```
jpn.words
      is       the      that        to        it        in    people
      29        17        14        14        13        10        10
     kill       and     death government       not        of
penalty
       8         7         7         7         7         7         7
       a        or        by         i      also        if
       6         6         5         5         4         4
```

９．散布図やグラフを書く

- 散布図

```
#  jpn.bodyのセンテンスの長さ（それぞれのセンテンスの単語数）を散布図にする。
# まず、jpn.bodyのそれぞれのセンテンスの単語数を出す関数は以下。
```

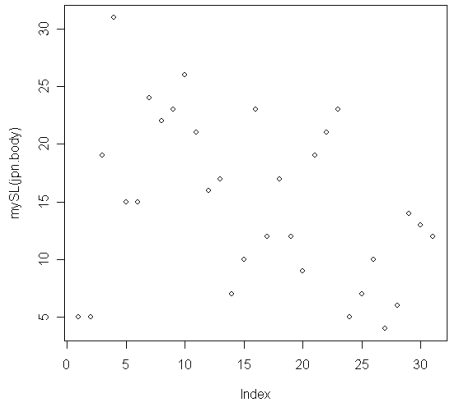
```
mySL <- function(a) {
       x <- strsplit(a, "¥¥W")
       y <- 0
       for (i in 1:length(x)) {
       y[i] <- length(x[[i]])
       }
       print(y)  # myASLとの違いはここのみ。yの平均を出さずにプリント。
```

```
}
```

```
> plot(mySL(jpn.body))  # plot で、散布図を描く。
```

(出力例)

```
 [1]  5  5 19 31 15 15 24 22 23 26 21 16 17  7 10 23 12 17 12  9 19 21
23  5  7
[26] 10  4  6 14 13 12
```



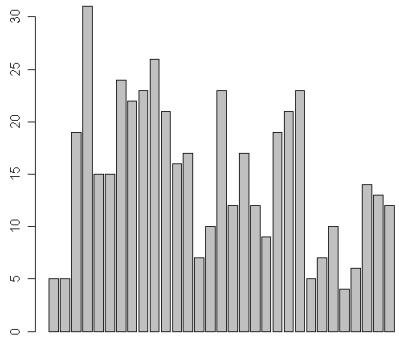
- 棒グラフ

jpn.body のセンテンスの長さを棒グラフにする。

```
> barplot(mySL(jpn.body))  # barplot で、棒グラフを描く。
```

(出力例)

```
 [1]  5  5 19 31 15 15 24 22 23 26 21 16 17  7 10 23 12 17 12  9 19 21
23  5  7
[26] 10  4  6 14 13 12
```



１０．　２gram とその頻度表を出す

- データの準備

```
# NICE から、学習者または母語話者の作文のみを取り出す関数は以下。
# これを R Console に貼り付ける。
myData <- function(){
       lines.tmp <- scan(choose.files(), what="char", sep="¥n")
         #ファイルを選択。
       data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
         #*で始まる行のみ。
       body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
         #行頭のタグを削除。
       body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
}

> jpn.body <- myData()  #画面に入力すると、ファイルを選択する窓が現われる。
    #JPN001.txt を選択し､myData で本文のみを取り出した結果を jpn.body に代入。

(出力例)
Read 121 items

> jpn.body
 [1] "Does death penalty really need?"
 [2] "Does death penalty really need?"
 [3] "I often hear that death penalty is reasonable for people whose
family member or friend is killed by someone."
（以下省略）
```

- 2 gram を出す関数

```
# 2 gram とは、2 語の連なりのこと。
# “I went to school yesterday” の場合、”I went”, “went to”, “to school”,
“school yesterday”が 2gram。
#まずは以下の関数を R Console に貼り付ける。

my2gram <- function(a){
       tmp.1 <- unlist(strsplit(a, "¥¥W"))   #単語をバラバラにする。
```

```
        for (i in 1:(length(tmp.1)-1)) {
            #i が 1 から (tmp の総語数-1)まで繰り返し。
                cat(tmp.1[i], tmp.1[(i+1)], "¥n")
                  #tmp の i 番目と i+1 番目をくっつける。
                }
        }
```

```
> my2gram(jpn.body)  #jpn.body に対し、関数 my2gram を実行。
```

```
(出力例)
Does death
death penalty
penalty really
really need
 (中略)
 hand
  I
(以下省略)
```

```
#上記結果を観察すると、1 語しか含まれない行が存在。
#元のテキストと比較すると、記号類と空白が重なっている場所でこのような現象が起こる。
#つまり、記号類で分割し、スペースが要素の 1 つとなっている。
#strsplit(a, "¥¥W")を strsplit(a, "¥¥W+")に変更し、ワード構成文字以外 2 つ
以上で分割。
```

```
my2gram <- function(a){
        tmp.1 <- unlist(strsplit(a, "¥¥W+"))
          #ワード構成文字以外 2 つ以上で分割。
        for (i in 1:(length(tmp.1)-1)) {
          #i が 1 から (tmp の総語数-1)まで繰り返し。
                cat(tmp.1[i], tmp.1[(i+1)], "¥n")
                  #tmp の i 番目と i+1 番目をくっつける。
                }
        }
```

```
> my2gram(jpn.body)
```

(出力例)
```
Does death
death penalty
penalty really
really need
need Does
```
(以下省略)

- 結果の保存と頻度表作成

#結果を 2gram.txt に保存する。まず、下記の関数を実行してみよう。

```
my2gram <- function(a){
       tmp.1 <- unlist(strsplit(a, "¥¥W+"))
       for (i in 1:(length(tmp.1)-1)) {
              cat(tmp.1[i], tmp.1[(i+1)], "¥n", file="2gram.txt")
                #結果を 2gram.txt に保存。
              }
       }
```

```
> my2gram(jpn.body)
```
#結果は 2gram.txt に書き出されているので、コンソール画面上には何も出ない。
#ところが、2gram.txt を対象に table で頻度表を作成すると以下。
```
> table(scan(choose.files(), what="char", sep="¥n"))
```
#2gram.txt を選択

(出力例)
```
Read 1 item

of it
    1
```

#結果が上書きされているため、最後の 2gram のみ表示
#そこで、file="2gram.txt"の後ろに、append=T（追加する）を加える。

```
my2gram <- function(a){
       tmp.1 <- unlist(strsplit(a, "¥¥W+"))
       for (i in 1:(length(tmp.1)-1)) {
```

```
            cat(tmp.1[i],  tmp.1[(i+1)],  "¥n",  file="2gram.txt",
      append=T)
            #結果をファイル 2gram.txt に保存。
            #上書きしないというオプション append=T をつける。
}
}

#2gram.txt を一旦削除し、もう一度 my2gram で jpn.body を処理。
>  my2gram(jpn.body)
> table(scan(choose.files(), what="char", sep="¥n"))
#2gram.txt を選択し、頻度表を作成

(出力例)
Read 437 items

     a government             a king           a person             a
point
               1                1                2                1
      a prisoner          abuse his        accepted to           adult
is
               1                1                1                1
(以下省略)

# 以下の関数では、結果を書き込むファイルを選択できる。
#存在しなしファイルを指定すると、「このファイルは存在しません。作成しますか？」と
出る。
# →「はい」をクリック

my2gram <- function(a){
      output.file <- choose.files()
      #結果を保存するファイルを選択（例：2gram_kekka.txt）。
      tmp.1 <- unlist(strsplit(a, "¥¥W+"))
        #ワード構成文字以外 2 つ以上で分割。
      for (i in 1:(length(tmp.1)-1)) {
        #i が 1 から (tmp の総語数-1)まで繰り返し。
      cat(tmp.1[i], tmp.1[(i+1)], "¥n", file=output.file, append=T)
            #tmp の i 番目と i+1 番目をくっつける。
            #指定したファイルに結果を保存。上書きなし。
```

```
        }
    }
```

- 2gram が何種類あるか

```
> length(unique(scan(choose.files(), what="char", sep="¥n")))
  #2gram の結果を保存したファイルを選択（例：2gram_kekka.txt）。
```

(出力例)

```
Read 437 items
[1] 354  #354 種類ある。
```

- 頻度上位の 2gram のみ表示

```
# 以下の関数 myTop で、頻度上位の 2gram のみ表示する。

myTop <- function(a){  # a で、何位まで欲しいか指定。
  tmp.n <- scan(choose.files(), what="char", sep="¥n")
  #2gram の結果を保存したファイルを指定（例：2gram_kekka.txt）。
  head(sort(table(tmp.n), decreasing = T), a)
    #降順にソートし、上位 a 位まで出す。
  }

> myTop(10)  #10 位まで表示させる。ファイルを指定する。
```

(出力例)

```
Read 437 items
tmp.n
       it is   death penalty         in the  moral standard       not
kill
           8             7             4             4             4
   that death   against moral         and we     Does death      hear
that
           4             3             3             3             3
```

１１．χ^2検定

(事例 1)

#LOB コーパス中の however の生起位置を調べたところ、次のような結果になった。

文頭	109
文中	347
文末	8

#この分布は偶然かどうか、χ^2検定を行う。

```
> however.data <- c(109, 347, 8)
    #however の頻度を however.data の要素に入れる。
> chisq.test(however.data)  #χ2検定を行う。

--------------------------結果の出力-----------------------
       Chi-squared test for given probabilities

data:  however.data
X-squared = 391.7371, df = 2, p-value < 2.2e-16
-----------------------------------------------------------------
```

#2.2e-16 意味は、2.2×10^{-6} （つまりほとんどゼロ）。

（結果の解釈例）

LOB コーパス中の however の生起位置には偏りがあり、文中が最も多い。

(事例２)

#イギリス英語とアメリカ英語で、 therefore の生起位置に違いがあるか調べる。

位置	イギリス	アメリカ
文頭	15 ①	38 ③
文中	96 ②	53 ④

#この分布は偶然かどうか、χ^2検定を行う。

```
therefore.data <- matrix(c(15,96,38,53), nrow=2, ncol=2)
                #matrix で行例作成。nrow は行数、ncol は列数。この場合は 2×2。

> therefore.data  #中身の確認。
```

```
    [,1] [,2]
[1,]   15   38
[2,]   96   53

> chisq.test(therefore.data)

--------------------------結果の出力-----------------------
       Pearson's Chi-squared test with Yates' continuity correction

data:  therefore.data
X-squared = 19.1788, df = 1, p-value = 1.190e-05
-------------------------------------------------------------------
```

（結果の解釈例）
　イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある（どこに有意差があるかは、残差分析を行う必要あり）。

１２．パッケージのインストール

- **方法１**

1. 「options()」 に CRAN の URL を設定し、関数「install.packages()」でパッケージを指定。
   # 例：パッケージ「zipfR」をインストールする場合。

   ```
   > options(CRAN="http://cran.r-project.org")
   > install.packages("zipfR")
   ```

2. 「--- このセッションで使うために、CRAN のミラーサイトを選んでください ---」とメッセージが現われ、ミラーサイトの選択ウィンドが開くので、Japan (Tsukuba)辺りを選択。
3. 「パッケージ 'zipfR' は無事に開封され、MD5 サムもチェックされました」とメッセージが現われれば、完了。

- **方法２**

1. GUI メニューから、「パッケージ」→「パッケージのインストール」を選択。
2. パッケージ選択窓が開くので、インストールしたいパッケージを選択 (例：zipfR)。
3. CRAN のミラーサイト選択窓が開くので、Japan (Tsukuba)辺りを選択。
4. 「パッケージ 'zipfR' は無事に開封され、MD5 サムもチェックされました」とメッ

セージが現われれば、完了。

１３．既存のスクリプトの利用

● スクリプトの保存

#他の人が書いてくれた関数をコピーし、テキストエディタに貼り付け、拡張子 R で保存。
#例）杉浦先生が書いた以下の関数をコピーし、my4gram.R と名前を付けて保存。

```
my4gram <- function(a){
output.file <- choose.files()
tmp.1 <- unlist(strsplit(a, "¥¥W+"))
for (i in 1:(length(tmp.1)-3)) {
cat(tmp.1[i], tmp.1[(i+1)], tmp.1[(i+2)], tmp.1[(i+3)], "¥n",
file=output.file, append=T)
}
}
```

● スクリプトの読み込み

(方法 1)
source(“関数ファイルのフルパス”)　#関数を読み込む。
例）source("C:/Documents and Settings/My Documents/LC2009/my4gram.R")

(方法 2)
#GUI メニューより、「ファイル」→「ディレクトリの変更」を選択し、関数ファイルのある場所を指定し、source(“ファイル名”) で関数を読み込む。
例）source(“my4gram.R”)

(方法 3)
source(choose.files())　#ファイルの選択窓が開くので、目的のファイルを選ぶ。

● スクリプトの実行

#例）jpn.body に対し、4gram を出してくれる my4gram.R を実行。

```
> source("my4gram.R")  #my4gram.R を読み込む。
> my4gram(jpn.body)   # my4gram.R の引数 a に jpn.body を渡す。
```

結果を書き込むファイルの選択窓が現われるので、例えば kekka.txt と入力し、「開く」。
#「このファイルは存在しません。作成しますか？」とメッセージが出るので、「はい」。
処理結果が kekka.txt に書き込まれる。

```
--------------kekka.txt----------------------------
Does death penalty really
death penalty really need
penalty really need Does
really need Does death
（以下省略）
-----------------------------------------------------
```

【演習：my4gram.R を my5gram.R にしてみる】

例：my5gram.R

```
-----------------------------------
my5gram <- function(a){
output.file <- choose.files()
tmp.1 <- unlist(strsplit(a, "¥¥W+"))
for (i in 1:(length(tmp.1)-4)) {  # for ループの回数が一回減り、
cat(tmp.1[i], tmp.1[(i+1)], tmp.1[(i+2)], tmp.1[(i+3)],tmp.1[(i+4)],
"¥n",
 # cat の要素数が１つ増える。
file=output.file, append=T)
}
}
-------------------------------------
```

１４．群馬大学青木繁伸先生のサイトの利用

- χ^2 検定と残差分析

#青木先生の作った関数を利用し、χ^2 検定後の残差分析をやってみよう。

(データ)
イギリス英語とアメリカ英語で、 therefore の生起位置に違いがあるか調べる。

位置	イギリス	アメリカ
文頭	15 ①	38 ③
文中	96 ②	53 ④

(手順)

１．青木先生のサイト（http://aoki2.si.gunma-u.ac.jp/R/）に行く。

２．「度数に関する検定」の下の「カイ二乗分布を使用する独立性の検定と残差分析」をクリック。

３．指示に従い以下の 1 行をコピーし，R コンソールにペーストする。

source("http://aoki2.si.gunma-u.ac.jp/R/src/my-chisq-test.R", encoding="euc-jp")」

４．分析したい数値を関数に送り、結果を変数 a に代入。

```
> (a <- my.chisq.test(matrix(c(15,96,38,53), nrow=2, ncol=2)))
```

----------結果の出力----------

カイ二乗分布を用いる独立性の検定（残差分析）

```
data:  matrix(c(15, 96, 38, 53), nrow = 2, ncol = 2)
X-squared = 20.6124, df = 1, p-value = 5.623e-06
```

```
# 自動的には残差分析まで出力してくれない。以下を実行。
> summary(a)
```

----------結果の出力----------

調整された残差

```
       [,1]   [,2]
[1,] -4.5401  4.5401
[2,]  4.5401 -4.5401
```

p 値

```
         [,1]      [,2]
[1,] 5.6231e-06 5.6231e-06
[2,] 5.6231e-06 5.6231e-06
```

（結果の解釈例）
　カイ二乗検定の結果より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。残差分析の結果より、イギリス英語では、アメリカ英語に比べて、therefore を文中で用いる頻度が有意に高く、文頭で用いる頻度が有意に低いことが分かる。

● 対数尤度比（G スコア）
#言語データは偏りがあり、カイ二乗分布に近似させることには無理があるため、対数尤度比を利用する方が望ましいという指摘がある。

(データ)
イギリス英語とアメリカ英語で、 therefore の生起位置に違いがあるか調べる。

位置	イギリス	アメリカ
文頭	15 ①	38 ③
文中	96 ②	53 ④

(手順)
１．青木先生のサイト（http://aoki2.si.gunma-u.ac.jp/R/）に行く。
２．「度数に関する検定」の下の「対数尤度比（G2）に基づく独立性の検定」をクリック
３．指示に従い以下の 1 行をコピーし，R コンソールにペーストする。

```
   source("http://aoki2.si.gunma-u.ac.jp/R/src/G2.R",
encoding="euc-jp")
```

４．分析したい数値を関数に送り、結果を変数 a に代入

```
> G2(matrix(c(15,96,38,53), nrow=2, ncol=2))
```

```
---------------------------結果の出力-----------------------
        対数尤度比に基づく独立性の検定（G-squared test）

data:  matrix(c(15, 96, 38, 53), nrow = 2, ncol = 2)
G-squared = 20.9249, df = 1, p-value = 4.776e-06
------------------------------------------------------------------
```

（結果の解釈例）
p< .001 より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。
注) どこに有意差があるかは教えてくれない。