```
#（2024年版）

################################
################################
####### [LIST] リスト ##########
################################
################################

#-------------------------
# [LIST.1] リストの生成
#-------------------------

# c関数で生成されるアトミックベクトルは、
# 全て同じ型であることが前提
a = c(13,TRUE,"hello") #異なる型のものを入れると強制的に同一の型へと変換
#[1] "13"    "TRUE"  "hello"

# aの型は「character」
typeof(a)
#[1] "character"

# list関数でつくられるリストであれば、
# 異なる型を1つの変数にグループとしてまとめることができる。
a = list(13,TRUE,"hello")
#[[1]]
#[1] 13
#[[2]]
#[1] TRUE
#[[3]]
#[1] "hello"

# aの型とクラスはともに「リスト」
typeof(a)
#[1] "list"
class(a)
#[1] "list"

# 個々の要素の型はtypeof([[n]])でわかる。
#  （R.2で詳しく説明します）
typeof(a[[1]]);typeof(a[[2]]);typeof(a[[3]])
#[1] "double"
#[1] "logical"
#[1] "character"



#-------------------------------
# [LIST.2] リスト[[n]]とリスト[n]
#-------------------------------

a = list(13,TRUE,"hello")
```

```
# リストのn番目の要素はリスト[[n]]で表します。
a[[1]];a[[2]];a[[3]]
#[1] 13
#[1] TRUE
#[1] "hello"

# リストのn番目の要素がベクトルの場合、
# m番目のサブ要素は[[n]][m]で表します。
a[[1]][1];a[[2]][1];a[[3]][1]
#[1] 11
#[1] TRUE
#[1] "hello"

# リスト[n]は、リスト[[n]]を要素として持つ「サイズ1」のリストを返します。
a[1]
#[[1]]
#[1] 13

# 以下の違いに注意。
typeof(a[[1]])
#[1] "double"
typeof(a[1])
#[1] "list"
typeof(a[1][[1]])
#[1] "double"


# リストの要素が全てアトミックベクトルの例
a = list(11:13,c(T,F,F,F),"good-bye")
#[[1]]
#[1] 11 12 13
#[[2]]
#[1]  TRUE FALSE FALSE FALSE
#[[3]]
#[1] "good-bye"

a[[1]][2];a[[2]][4]
#[1] 12
#[1] FALSE

#aの三番目の要素はアトミックベクトル
a[[3]]
#[1] "good-bye"

# a[3]は、a[[3]]を要素に持つサイズ1のリスト（復習）
a3 = a[3]
#[[1]]
#[1] "good-bye"

typeof(a3);length(a3)
#[1] "list"
```

```
#[1] 1


# a[1]は、11:13（a[[1]]）を要素に持つサイズ1のリスト（復習）
a1 = a[1]
#[[1]]
#[1] 11 12 13

typeof(a1);length(a1)
#[1] "list"
#[1] 1


# リストはリストを要素に加えることができます。
a = list(list(T,c("hello","good-bye")),c(T,F,F,F),101:200)

# リストのリストは[[p]][[q]]..という形で掘っていく
a[[1]] #aの第一要素もリスト構造
#[[1]]
#[1] TRUE
#[[2]]
#[1] "hello"    "good-bye"

# aの第１要素の第２要素はアトミックベクトル
a[[1]][[2]]
#[1] "hello"    "good-bye"

# "good-bye"は
# a（リスト）の第１要素（リスト）の第２要素（ベクトル）の第２要素
a[[1]][[2]][2]
#[1] "good-bye"



# くどいですが、再び、リストとアトミックベクトルの違いの復習

a = c("hello","good-bye")
b = list("hello","good-bye")

# "hello"の参照は？
a[1];b[[1]][1]
#[1] "hello"
#[1] "hello"

# "good-bye"の参照は？
a[2];b[[2]][1]
#[1] "good-bye"
#[1] "good-bye"



#--------------------------
```

```
# [LIST.3] $記法（名前属性）
#--------------------------


# リストの各要素には名前属性をつけることができます。

chubu = list(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名前
             pop = c(716,211,186,379), #人口
             order = c(4,17,22,10), #人口順位
             capital = c("nagoya","gifu","mie","shizuoka"), #県庁所在
地
             shinkansen = c(T,T,F,T)) #新幹線通過の有無

# 人口にアクセスするには次の2通り存在する
chubu[[2]] #二重括弧記法
#[1] 716 211 186 379
chubu$pop #$記法
#[1] 716 211 186 379

#愛知県の県庁所在地は、
chubu$capital[1]
#[1] "nagoya"

#岐阜県の人口順位は
chubu$order[2]
#[1] 17

#三重県の新幹線通過の有無は、
chubu$shinkansen[3]
#[1] FALSE

#静岡県の人口は
chubu$pop[4]
#[1] 379



#-----------------------------------
#（LIST.4）リストを出力する関数の例
#-----------------------------------


#先週の授業の例
#n個のサイコロを振って、総和を得るプログラム
roll.n = function(n){
  dice = 1:6
  dice2 = sample(dice, size=n, replace = TRUE)
  # dice2 = sample(dice,n,T) #こちらでも可
  sum(dice2)
```

```
}

# サイコロを3回振った総和の1000サンプルのベクトルです。
sample = replicate(1000,roll.n(3))
#[1] 11 11 10  9 12 10 10 12 10  9  9 13 13 10 12 14
#[17] 16  5  8 12 10 12  5 15  9  6  3 16  9 10  5  9
#省略
#[977] 10  6 12  9 11  8 15 11 11  6  8  9 11 14 11 11
#[993] 10 17  8 10 12  8 13  7

# ヒストグラムを計算し、Plotsパネルにグラフを出力します。
hi = hist(sample,breaks=1:20)

# hiはhistogramクラスのリストです。
typeof(hi);class(hi)
#[1] "list"
#[1] "histogram"

# 右上のEnvironmentパネルで「hi」を開くと
# 6つの名前属性（$breraks,$counts,$density,$mids,$xname,$equidist）
# の存在が確認できます。

# 総和サンプルの一覧
hi$breaks
#[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
hi[[1]] #リストなので二重括弧を使うこともできます。
#[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

#個々のサンプル数（例です）
hi$counts #hi[[2]]でも同じです。
#[1]   0   4  10  26  54  73 111 109 124 129 106 101  60  43  32  16
2   0   0



#---------------------------------
#（LIST.5）論理値記法
#---------------------------------

# 簡単なベクトルを例に論理値サーチの方法を見ていきます。
# まず1から20までの整数列をつくります。
n = 1:20
#[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

# 同じサイズの論理値のベクトルを使うと特定の要素だけ取り出すことができます。
# こちらは奇数部分だけを取り出す例
n[c(T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F)]
#[1]  1  3  5  7  9 11 13 15 17 19
n[c(T,F)] #リサイクル規則
#[1]  1  3  5  7  9 11 13 15 17 19
```

```
n[c(T,F,F)] #リサイクル規則
#[1]  1  4  7 10 13 16 19

# c(T,F,..)というベクトルは以下の方法でも作れます。
n %% 2 == 1
#[1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
TRUE FALSE  TRUE FALSE
#[15]  TRUE FALSE  TRUE FALSE  TRUE FALSE

# これを使えば、よりスマートに、特定の要素を取り出すことが可能です。
n[n %% 2 == 1]
#[1]  1  3  5  7  9 11 13 15 17 19
n[n %% 3 == 1]
#[1]  1  4  7 10 13 16 19

# 1000から100個のサンプルを抽出し、そのうち10で割り切れるもののみを取り出す。
n = sample(1000,100)
n[n %% 10 == 0]
#[1] 610 920 890 200 470 940

#ここからリストに戻ります。
roll.n = function(n){
  dice = 1:6
  dice2 = sample(dice,n,T)
  sum(dice2)
}

sample = replicate(1000,roll.n(3))

#breaksは区切りのライン
#2.5~3.5, 3.5~4.5, 4.5~5.5, ...17.5~18.5をサンプリング
hi = hist(sample,breaks=2.5:18.5)

# hi$XXXはベクトルのため、同じ探索方法が適用できます。

# hi$midsは、各ビンの中央のサンプル値
hi$mids
#[1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
hi$counts
#[1]   7  10  35  53  51 105 114 106 138 119  97  66  60
#[14]  23  15   1

# 総和10のサンプル数は？
#  （後述の論理値サーチを参照）
hi$counts[hi$mids==10]
# [1] 106

# 総和10以上の各サンプル数は？
hi$counts[hi$mids>=10]
#[1] 106 138 119  97  66  60  23  15   1
```

```
# 総和10以上のサンプル数の総和は、
sum(hi$counts[hi$mids>=10]) #欠損値があるとNAになります
#[1] 625

# 総和10以上の確率密度は？
sum(hi$counts[hi$mids>=10])/sum(hi$counts)
#[1] 0.625



####################################
####################################
####### [DF] データフレーム #########
####################################
####################################


#----------------------------------
# [DF.1] データフレームの生成
#----------------------------------

# いよいよデータフレームを扱います。
# データフレームはリストの一種ですが、
# よりデータ分析に特化したリストと考えることができます。

# （LIST.3）で作った中部地方のリストのコンストラクタの関数部分を、
# listからdata.frameに変更します。

chubu = data.frame(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名
前
             pop = c(716,211,186,379), #人口
             order = c(4,17,22,10), #人口順位
             capital = c("nagoya","gifu","mie","shizuoka"), #県庁所在
地
             shinkansen = c(T,T,F,T)) #新幹線通過の有無

# chubuの中身を見ると、データが綺麗に整列されています。
chubu
#      name pop order  capital shinkansen
#1    AICHI 716     4   nagoya       TRUE
#2     GIFU 211    17     gifu       TRUE
#3      MIE 186    22      mie      FALSE
#4 SHIZUOKA 379    10 shizuoka       TRUE

# 型はリストのまま、クラスがdata.frameとなる。
typeof(chubu);class(chubu)
#[1] "list"
#[1] "data.frame"
```

```
# DF[n]は、n番目の要素をデータフレーム形式で返します。
chubu[2]
#pop
#1 716
#2 211
#3 186
#4 379

chubu[4]
#capital
#1   nagoya
#2     gifu
#3      mie
#4 shizuoka


# データフレームは、全ての要素がアトミックベクトルで
# かつ、それらの要素数が同一でないとエラーが返されます。
# 例えば、以下の場合、shinkansenの要素数のみが3で揃わないためエラーとなります。
chubu = data.frame(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名
前
                   pop = c(716,211,186,379), #人口
                   order = c(4,17,22,10), #人口順位
                   capital = c("nagoya","gifu","mie","shizuoka"), #
県庁所在地
                   shinkansen = c(T,T,F)) #新幹線通過の有無
#Error in data.frame(name = c("AICHI", "GIFU", "MIE", "SHIZUOKA"),
pop = c(716,
#:arguments imply differing number of rows: 4, 3




#-------------------------
# [DF.2] ij記法 （DF[i,j]）
#-------------------------

chubu = data.frame(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名
前
                   pop = c(716,211,186,379), #人口
                   order = c(4,17,22,10), #人口順位
                   capital = c("nagoya","gifu","mie","shizuoka"), #
県庁所在地
                   shinkansen = c(T,T,F,T)) #新幹線通過の有無
chubu
#      name pop order  capital shinkansen
#1    AICHI 716     4   nagoya       TRUE
#2     GIFU 211    17     gifu       TRUE
```

```
#3      MIE 186    22      mie      FALSE
#4 SHIZUOKA 379    10 shizuoka       TRUE

# DF[i,j]で、データフレームを行列と見立てた時の
# i行目j列目の要素を取り出すことができます。

# 三重県の人口順位（3行3列目）
chubu[3,3] #対応するベクトル要素を返します。
#[1] 22

# 名前属性を使った$記法も、リスト要素の記法も使えます。
chubu$order[3];chubu[[3]][3]
#[1] 22
#[1] 22

# ij記法の拡張
# jの要素数が複数の場合、データフレームを返します！！
chubu[3,c(1,3)] #三重の県名と人口順位（3行の1列目と3列目）
#  name order
#3  MIE    22

chubu[1:3,c(1,5)] #静岡以外の名前と新幹線情報
#   name shinkansen
#1 AICHI       TRUE
#2  GIFU       TRUE
#3   MIE      FALSE

chubu[-4,c(1,5)] #上と同じです。（4行目を除外）
#   name shinkansen
#1 AICHI       TRUE
#2  GIFU       TRUE
#3   MIE      FALSE

chubu[3,] #三重県の全て
#  name pop order capital shinkansen
#3  MIE 186    22     mie      FALSE

chubu[,4] #全ての県の県庁所在地
#[1] "nagoya"   "gifu"     "mie"      "shizuoka"
# jの要素数が1つなので、ベクトルが返ります。

# !! 1つの行を選択すると,データフレームを返す一方で、
# !! 1つの列を選択すると,アトミックベクトルを返すことに注意

# 順番の入れ替え
chubu[c(2,1,3,4),]
#      name pop order  capital shinkansen
#2     GIFU 211    17     gifu       TRUE
#1    AICHI 716     4   nagoya       TRUE
#3      MIE 186    22      mie      FALSE
```

```
#4 SHIZUOKA 379    10 shizuoka       TRUE

# 列指定で順番の入れ替え
chubu[c(2,1,3,4),2] #人口を愛知・岐阜・三重・静岡の順で
#[1] 211 716 186 379
# 列のサイズが1なので、やはりベクトルが返ります。

# 行も列も順番の入れ替え
chubu[c(2,1,3,4),c(4,1)]
#capital     name
#2     gifu     GIFU
#1   nagoya    AICHI
#3      mie      MIE
#4 shizuoka SHIZUOKA
# 列のサイズが2なので、データフレームで返します。



#---------------------------------
# [DF.3] 名前による指定、$記法
#---------------------------------

chubu = data.frame(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名
前
                   pop = c(716,211,186,379), #人口
                   order = c(4,17,22,10), #人口順位
                   capital = c("nagoya","gifu","mie","shizuoka"), #
県庁所在地
                   shinkansen = c(T,T,F,T)) #新幹線通過の有無

# 列の名前属性を、ij記法のjに直接に指定することができます。

# jのサイズが複数の場合、データフレームとして返ります。
chubu[1,c("name","pop")] #愛知県の名前と人口
#   name pop
#1 AICHI 716
chubu[1,c("pop","name")] #名前を入れ替えると順番も変わる
#  pop  name
#1 716 AICHI
chubu[,c("name","pop")] #全ての県の名前と人口
#      name pop
#1    AICHI 716
#2     GIFU 211
#3      MIE 186
#4 SHIZUOKA 379

# jのサイズが1の場合、ベクトルが返ることに注意。
chubu[,c("pop")] #全ての県の人口
#[1] 716 211 186 379
```

```
# データフレームはリストの一種ですので、
# [LIST.3]で紹介した$記法を適用できます。

chubu$capital #全ての県の県庁所在地、ベクトルを返します。
#[1] "nagoya"   "gifu"     "mie"      "shizuoka"

chubu$shinkansen #全ての県の新幹線状況（ベクトル）
#[1]  TRUE  TRUE FALSE  TRUE

chubu$pop #全ての県の人口（ベクトル）
#[1] 716 211 186 379

sum(chubu$pop) #総和
#[1] 1492

mean(chubu$pop) #平均
#[1] 373

# ほとんどの関数は、引数にアトミックベクトルをとりますが、
# リストを引数にとることはできません。
# そのため、データフレームのデータをベクトルに変換する
# $記法は、 非常に重宝します。


mean(chubu[2]) #エラー（chubu[2]はベクトルではなくリスト!!）
#[1] NA
#Warning message:
#  In mean.default(chubu[2]) :
#  argument is not numeric or logical: returning NA

mean(list(c(716,211,186,379))) #同じくエラー
#[1] NA
#Warning message:
#  In mean.default(chubu[2]) :
#  argument is not numeric or logical: returning NA

# 二重括弧でベクトルに変換すれば、一般的な関数の引数にとれます。
mean(chubu[[2]])
#[1] 373




#--------------------------
# [DF.4] 論理値による探索
#--------------------------
```

```
chubu = data.frame(name = c("AICHI","GIFU","MIE","SHIZUOKA"), #県の名前
                   pop = c(716,211,186,379), #人口
                   order = c(4,17,22,10), #人口順位
                   capital = c("nagoya","gifu","mie","shizuoka"), #県庁所在地
                   shinkansen = c(T,T,F,T)) #新幹線通過の有無

# (LIST.5)で扱った論理値による探索は
# データフレームで大活躍します。


chubu[1,c(T,T,F,F,F)] #愛知県の名前と人口のみ
#   name pop
#1 AICHI 716
chubu[c(T,F,F,F),c(1,2)] #同じです。
#   name pop
#1 AICHI 716


#  人口順位のみをベクトルとして取り出す
chubu$order
#[1]  4 17 22 10

#  人口順位10番以内の県をサーチ（真偽値ベクトルが返ります）
chubu$order<=10
#[1]  TRUE FALSE FALSE  TRUE

#  条件を満たす行をデータフレームとして取り出す
chubu[chubu$order<=10,]
#      name pop order  capital shinkansen
#1    AICHI 716     4   nagoya       TRUE
#4 SHIZUOKA 379    10 shizuoka       TRUE

# 条件を満たす県をベクトルで取り出す
# 以下は、人口トップ10の県を取り出す例
chubu[chubu$order<=10,1]
#[1] "AICHI"    "SHIZUOKA"
chubu[[1]][chubu$order<=10]
#[1] "AICHI"    "SHIZUOKA"
chubu$name[chubu$order<=10]
#[1] "AICHI"    "SHIZUOKA"


# 新幹線の通らない県の人口順位
chubu$order[chubu$shinkansen==F]
#[1] 22

# 新幹線の通る県の人口順位
```

```
chubu$order[chubu$shinkansen==T]
#[1]  4 17 10




#-----------------------------------------------
# [DF.5] CSVファイルからデータフレームを読み取り
#-----------------------------------------------


# 以下から、メジャーリーグの各年の成績データが入手できます。
# https://baseballsavant.mlb.com/leaderboard

# 以下は、上記から2021〜2022年の成績をCSVとしてDLしたものです。
# バッターの成績：2021 - 2022
# http://lab.kenrikodaka.com/_download/class/2023_AppliedMedia/
mlb.b.2021-22.csv
# ピッチャーの成績：2021 - 2022
# http://lab.kenrikodaka.com/_download/class/2023_AppliedMedia/
mlb.p.2021-22.csv


# CSVファイルは以下のようなフォーマットになっています。
# 一行目に名前属性、二行目以降に各データが並んでいます。
# last_name,...,year,player_age,...home_run,...
# Frazier,..,2021,29,...,5,...
# Altuve,...,2021,31,...,31,...
# Gallo,....,2021,38,...,38,...
# Sano,.....,2021,30,...,30,...
# ...

# last_name（ラストネーム）,first_name（ファーストネーム）,
# player_id（id）,year（成績年）,player_age（年齢）,
# ab（打席数）,hit（安打数）,single（単打）,double（二塁打）,triple（三塁
打）,
# home_run（本塁打）,strikeout（三振）,walk（四球）,
# k_percent（三振率）,bb_parcent（四球率）,
# batting_avg（打率）,on_base_percent（出塁率）,on_base_plus_slg（OPS）
# wOBA（打撃貢献指標）, X（?）

## CSVファイルのインクルード
###（1）ローカルファイルから

# 作業ディレクトリを指定します。（以下は例です。）
setwd("/Users/kenrikodaka/Dropbox/DocClass/_2023/B3_情報処理応用")
# メニューからも指定できます（Session → ChooseDirectory）
```

```
# setwdをしておくと、データを読み込むときに楽になります。
# 以下のようにカレントディレクトリからの相対パスでデータを指定できます。
# read.csvを用いると、csvデータをデータフレームとして読み込むことができます。
datb = read.csv("_dat/mlb.b.2021-22.csv") #打撃成績
datp = read.csv("_dat/mlb.p.2021-22.csv") #投手成績


###（2）インターネットから直接

# バッターの成績：2021 - 2022
url_b = "http://lab.kenrikodaka.com/_download/class/
2023_AppliedMedia/mlb.b.2021-22.csv"
# ピッチャーの成績：2021 - 2022
url_p = "http://lab.kenrikodaka.com/_download/class/
2023_AppliedMedia/mlb.p.2021-22.csv"

datb = read.csv(url_b) #打撃成績
datp = read.csv(url_p) #投手成績


#-------------------------------------------
# [DF.6] 練習（MLBのデータから）
#-------------------------------------------


# 打撃成績データの最初の6行をちょっと出し
head(datb)

#last_name first_name player_id year player_age  ab  pa hit single
#1      Frazier       Adam    624428 2021         29 577 639 176
130
#2       Altuve       Jose    514888 2021         31 601 678 167
103
#3        Gallo       Joey    608336 2021         27 498 616  99
47
#4         Sano     Miguel    593934 2021         28 470 532 105
51
#5 Kiner-Falefa      Isiah    643396 2021         26 635 677 172
136
#6        Edman      Tommy    669242 2021         26 641 691 168
113
#double triple home_run strikeout walk k_percent bb_percent
batting_avg
#1     36      5        5        69   48      10.8        7.5
0.305
#2     32      1       31        91   66      13.4        9.7
0.278
#3     13      1       38       213  111      34.6       18.0
0.199
#4     24      0       30       183   59      34.4       11.1
0.223
```

```
#5     25      3        8        90   28      13.3        4.1
0.271
#6     41      3       11        95   38      13.7        5.5
0.262
#slg_percent on_base_percent on_base_plus_slg  woba  X
#1       0.411           0.368            0.779 0.341 NA
#2       0.489           0.350            0.839 0.357 NA
#3       0.458           0.351            0.809 0.348 NA
#4       0.466           0.312            0.778 0.332 NA
#5       0.357           0.312            0.669 0.293 NA
#6       0.387           0.308            0.695 0.301 NA

# 打率が3割を超えている選手を列挙
datb$last_name[datb$batting_avg>0.3]
#[1] "Frazier"      "Harper"       "Marte"        "Anderson"
"Soto"         "Castellanos"  "Brantley Jr."
#[8] "Riley"        "Reynolds"     "Gurriel"      "Turner"
"Guerrero Jr." "Freeman"      "Alvarez"
#[15] "Benintendi"   "Judge"        "Goldschmidt"  "Bogaerts"
"Lowe"         "McNeil"       "Abreu"
#[22] "Arraez"


# ホームランを40本以上売っている選手の名前を列挙
datb$last_name[datb$home_run>=40]
#[1] "Tatis Jr."    "Ohtani"       "Perez"        "Guerrero Jr."
"Semien"       "Schwarber"    "Judge"
#[8] "Alonso"

# ホームランを40本以上売っている選手の成績を抽出
# （データフレームとして出力されます）
datb[datb$home_run>=40,]
#last_name first_name player_id year player_age
#15     Tatis Jr.   Fernando    665487 2021         22
#27        Ohtani     Shohei    660271 2021         26
#110        Perez   Salvador    521692 2021         31
#125 Guerrero Jr.   Vladimir    665489 2021         22
#131       Semien     Marcus    543760 2021         30
#170    Schwarber       Kyle    656941 2022         29
#189        Judge      Aaron    592450 2022         30
#252       Alonso       Pete    624413 2022         27

# ホームランを40本以上の記録を列挙
# （アトミックベクトルでの出力）
datb$home_run[datb$home_run>=40]
#[1] 42 46 48 48 45 46 62 40

# ホームランを40本以上の選手の名前・成績年・本塁打数を
# データフレームで抽出
datb[datb$home_run>=40,c("last_name","year","home_run")]
#last_name year home_run
#15     Tatis Jr. 2021       42
```

```
#27        Ohtani 2021       46
#110        Perez 2021       48
#125 Guerrero Jr. 2021       48
#131       Semien 2021       45
#170    Schwarber 2022       46
#189        Judge 2022       62
#252       Alonso 2022       40


# ブール演算子
# A & B（AかつBが真のとき真）
# A | B（AまたはBが真のとき真）
# xor(A,B)（AとBのうち1つだけが真のとき真）
# !A（Aが偽のとき偽）
# any(A,B,C,...)（いずれかが真のとき真）
# all(A,B,C,...)（いずれも真のとき真）


# 再び、打率が3割を超えている選手を列挙
datb$last_name[datb$batting_avg>0.3]
#[1] "Frazier"      "Harper"       "Marte"        "Anderson"
"Soto"         "Castellanos"  "Brantley Jr."
#[8] "Riley"        "Reynolds"     "Gurriel"      "Turner"
"Guerrero Jr." "Freeman"      "Alvarez"
#[15] "Benintendi"   "Judge"        "Goldschmidt"  "Bogaerts"
"Lowe"         "McNeil"       "Abreu"
#[22] "Arraez"

# 2021年に限定（&は論理和）
bat2021 = datb$last_name[datb$year==2021 & datb$batting_avg>0.3]
#[1] "Frazier"      "Harper"       "Marte"        "Anderson"
"Soto"         "Castellanos"  "Brantley Jr."
#[8] "Riley"        "Reynolds"     "Gurriel"      "Turner"
"Guerrero Jr."

# 2022年に限定
bat2022 = datb$last_name[datb$year==2022 & datb$batting_avg>0.3]
#[1] "Freeman"     "Alvarez"     "Benintendi"  "Judge"
"Goldschmidt" "Bogaerts"    "Lowe"
#[8] "McNeil"      "Abreu"       "Arraez"



# 2年連続3割を超えた打者を探します。
# まずはintersectの使い方を覚えます。

# intersectは2つのベクトルの中の共通の要素を取り出します。
intersect(1:50,30:80)
#[1] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
# 2の倍数と3の倍数の共通要素（0～50）は6の倍数
intersect(seq(0,50,by=2),seq(0,50,by=3))
```

```
#[1]  0  6 12 18 24 30 36 42 48

# 2年連続の3割打者はいない
intersect(bat2021,bat2022)
# character(0)

# 2年連続、2割8部超えは8人（ジャッジ、ボガーツ、ゴールドシュミット他）
tmp2021 = datb$last_name[datb$year==2021 & datb$batting_avg>0.28]
tmp2022 = datb$last_name[datb$year==2022 & datb$batting_avg>0.28]
intersect(tmp2021,tmp2022)
#[1] "Marte"       "Judge"       "Bogaerts"    "Rosario"
"Goldschmidt" "Turner"      "Freeman"
#[8] "Bichette"


# 同様に、2年連続40本以上の選手は？
tmp2021 = datb$last_name[datb$year==2021 & datb$home_run>=40]
tmp2022 = datb$last_name[datb$year==2022 & datb$home_run>=40]
intersect(tmp2021,tmp2022)
# character(0) #（いない）

# 2年連続34本以上の選手は？
tmp2021 = datb$last_name[datb$year==2021 & datb$home_run>=34]
tmp2022 = datb$last_name[datb$year==2022 & datb$home_run>=34]
intersect(tmp2021,tmp2022)
# [1] "Ohtani" "Judge"  "Alonso" "Olson" #（4人いた）

# 関数化します。（BASICの復習）
getHomerunBatter = function(n){
  tmp2021 = datb$last_name[datb$year==2021 & datb$home_run>=n]
  tmp2022 = datb$last_name[datb$year==2022 & datb$home_run>=n]
  intersect(tmp2021,tmp2022)
}

# こんな感じで同じことができますね。
getHomerunBatter(40)
#character(0)
getHomerunBatter(39)
#[1] "Judge"
getHomerunBatter(38)
#[1] "Judge"
getHomerunBatter(37)
#[1] "Judge"  "Alonso"


# 打撃成績にも投手成績にもどちらにも顔を出している選手のIDは？
#  （規定打席と規定投球回数のいずれも満たしている選手です）
intersect(datb$player_id,datp$player_id)
#[1] 660271

#誰だ？
```

```
#名前、成績年、打席数
datb[datb$player_id==660271,c("last_name","year","ab")]
#last_name year ab
#27     Ohtani 2021 537
#216    Ohtani 2022 586

#名前、成績年、投球回数
datp[datp$player_id==660271,c("last_name","year","p_formatted_ip")]
#last_name year p_formatted_ip
#84     Ohtani 2021          130.1
#179    Ohtani 2022          166.0
#(大谷は2021年は規定投球回数満たしてないよね、、)
```