```
########################
#(1) 変数と代入 ########
########################

#コメントアウトは#を行頭につける

#変数aに1を代入
#イコールも矢印もどちらも使えます。
a<-1
a=1
#Rでは矢印が一般的ですが、この授業ではイコール記法を採用します。

#実行する行を選択し、「⌘RETURN」でコードを実行
#コンソールに以下のように実行結果が表示されます
#>[1] 1
a
#[1] 1

#ピリオドを変数の表記に使うことも可（先頭は不可）
A.a = 0.4; A.b = 0.8
A.a; A.b #セミコロンを使うと一行に複数の文を並べることができます。
#[1] 0.4
#[1] 0.8

#大文字と小文字は区別します
name<-1; Name<-2
name+1
#[1] 2


########################
#(２) 型 ###############
########################


#https://qiita.com/maruman029/items/365a2abcdaaf99b720be

a = 1 #整数はデフォルトではdouble型として扱われます
b = 0.2 #double型
c = TRUE #logical型（T、F表記も可能）
d = 1L #整数型（integer型）として扱いたい時はLをつける
e = "Hello" #character型

#データ型はtypeofで調べる
typeof(a);typeof(b);typeof(c);typeof(d);typeof(e)

#型変換にはas.xxxを使う
a = 1; typeof(a)
#[1] "double"
a = as.integer(a) #double型からinteger型に変換
typeof(a)

#numericはintegerとdoubleを合わせた型（mode関数で確認可能）
mode(a);mode(b)


###################################
```

```
## （３）ベクトルの生成 ############
####################################

##(3.1) ベクトルを生成する方法

# c(n1,n2,n3)
# cはcombineに由来

a=c(1,2,3)
#[1] 1 2 3
a=c(T,T,F)
#[1] T T F
a=c("T","T","F") #文字列の配列
#[1] "T" "T" "F"

# c(a:b)で一次元配列ベクトル（a,a+1,,b）を作成
a=c(101:105)
#[1] 101 102 103 104 105
a=c(9:1); # b<aのとき（a,a-1, .. b）
#[1] 9 8 7 6 5 4 3 2 1
a = c(-3:9999333)
length(a) #ベクトルの長さはlength関数で取得可能
#[1] 9999337

# c(a,b)の引数が非整数の場合がある。
# 初項a、公差1で、b以下のものを集めたベクトルを返す
a=c(2.3:4)
#[1] 2.3 3.3

# :記法の場合, cは省略可能
a=101:105
#[1] 101 102 103 104 105
a=9:1
#[1] 9 8 7 6 5 4 3 2 1
a=2.3:4
#[1] 2.3 3.3

a = seq(1,10,length=5) #1~10を5分割
#[1] 1.00  3.25  5.50  7.75 10.00
a= seq(1,10,by=2) #1から2ずつ増やして10を越える手前まで
#[1] 1 3 5 7 9
a = rep(1:3,times=3) #(1,2,3)を3回
#[1] 1 2 3 1 2 3 1 2 3
a = rep(1:3,length=5) #(1,2,3)を要素数5となるまで繰り返す
#[1] 1 2 3 1 2


# 無作為サンプリング関数（sample）を使う
a = sample(1:6,1) #サイコロを1回振る
#[1] 5
#[1] 3
a = sample(1:6,6) #1~6から無作為サンプルを6回（重複なし）
#[1] 4 6 5 1 3 2

a = sample(1:6,7) #エラー（母数よりサンプル数が多い）
a = sample(1:6,7,replace = TRUE) #replaceをTRUEにセットすると重複ありモードへ
# [1] 6 1 2 6 5 6 5
```

```
# 空のベクトルを作成
a=vector("integer") #integer型の空のベクトルを生成（中身は空）
#integer(0)
a=vector("logical")
#logical(0)
a=vector("integer",10) #引数2はベクトルの長さ（初期値は0）
#[1] 0 0 0 0 0 0 0 0 0 0
a=vector("logical",10) #logical型の場合、初期値はFALSE
#[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
##(3.2) ベクトルの結合

a = c(101:105)
a = c(a,201,202) #2つの要素を追加
#[1] 101 102 103 104 105 201 202

# append関数を使って結合する
a = c(101:105)
b = c(301:305)
append(a,b) #aとbを結合
#[1] 101 102 103 104 105 301 302 303 304 305
append(a,b,after=2) #3番目の要素から挿入
#[1] 101 102 301 302 303 304 305 103 104 105
```

```
##(3.3) ベクトルの要素抽出・置換

# n番目の要素を取り出す
a=101:105
a[3]
#[1] 103

#  n番目の要素を書き換える
a=101:105; a[3] = 3;
#[1] 101 102 3 104 105

a=101:105
a[c(2,4)] #2番目と4番目の要素を取り出してベクトルにする
# [1] 102 104
a[c(-2,-4)] #2番目と4番目の要素を取り除く
# [1] 101 103 105
a[4:8] #範囲外はNAを要素とする
# [1] 104 105  NA  NA  NA

# 特定の条件を満たす要素のみを取り出す時は
# which()関数で、引数に条件を記述する

a=101:105
a[which(a-1==100)] #a-1が100の要素のみを取り出す
# [1] 101

# 整数商は「%/%」、割った余りは「%%」
# 例えば, 17わる3は、5余り2
```

```
17 %/% 3; 17 %% 3
# [1] 5
# [1] 2
#これを使うと、、
a=101:105
a[which(a%%2==0)] #偶数の要素のみを取り出す
# [1] 102 104
a[which(a%/%103<1)] #103で割った余りが1より小さい
# [1] 101 102



##(3.4) その他

#（アトミック）ベクトルかどうかを確認する
a=2:5; is.vector(a)
#[1] TRUE

#リストはベクトル（リストは後に勉強します）
a=list(2:5,"R"); is.vector(a)
#[1] TRUE
#データフレームはベクトルでは無い（データフレームは後で勉強します）
a=data.frame(c(1,2),c("KODAKA","KENRI")); is.vector(a)
#[1] FALSE



#アトミックベクトルは一次元ベクトルとして値を格納
#（!!!）アトミックベクトルは一つの同じ型のデータしか格納できない!!
#（これこそがアトミックベクトルの定義ともいえます）
#アトミックベクトルの型はtypeofで調べる。

a=c(T,T,F); b=c(101:105)
typeof(a); typeof(b)
#[1] "logical"
#[1] "integer"

#  異なる型を入れた場合、適当な型に型変換されます。
#  以下で、Tと102は文字列型（character）に型変換されます。
a=c(T,102,"hello"); typeof(a)
#[1] "character"

a[1];a[2];a[3]
#[1] "TRUE"
#[1] "102"
#[1] "hello"

# 数値も要素1のベクトルです。
a=2;is.vector(a)
#[1] TRUE
a[1]
#[1] 2



####################################
## （３）ベクトルの演算 ############
####################################
```

```
dice=1:6
#[1] 1 2 3 4 5 6
dice-1
#[1] 0 1 2 3 4 5
dice/2
#[1] 0.5 1.0 1.5 2.0 2.5 3.0
dice * dice
#[1]  1  4  9 16 25 36
dice %% 3 #3で割った余り
#[1] 1 2 0 1 2 0

# （リサイクル規則）
# ベクトルのサイズが異なる場合の演算は、
# 短い方のベクトルの要素が繰り返し使われます。
x = c(2,2,3,3,4,4)
y = c(0.1,0.2)
z = x+y
#[1] 2.1 2.2 3.1 3.2 4.1 4.2
z = x*y
#[1] 0.2 0.4 0.3 0.6 0.4 0.8


##############################################
## （４）ベクトルに適用される関数 ############
##############################################

vec = 1:6
sum(vec) #総和
#[1] 21
mean(vec) #平均
#[1] 3.5
round(mean(vec))#四捨五入
#[1] 4


#ランダムサンプリング
sample(1:4, 2)
#[1] 2 4
#[1] 4 1

#上と同様（引数の名前をセットする）
sample(x=1:4,size=2)
sample(size=2,1:4)

#関数の引数の名前（および引数のデフォルト値）を調べる
args(sample)
#function (x, size, replace = FALSE, prob = NULL)

#replaceは同一の値のサンプリングを許可するか
sample(1:4, replace=TRUE)
#[1] 1 2 2 3
#[1] 4 3 3 2

#名前がなければ、2番目の引数はsizeが期待されるためエラー
sample(1:4, TRUE)
#Error in sample.int(length(x), size, replace, prob) :
#  invalid 'size' argument
```

```
#誤った引数の名前はエラーとなる
sample(1:4,saizu=2)
#Error in sample(1:4, saizu = 2) : unused argument (saizu = 2)



##############################################
## （５）配列 ############
##############################################


#一次元ベクトルを2行3列の配列に変換
a = array(1:6,dim=c(2,3))
#[,1] [,2] [,3]
#[1,]    1    3    5
#[2,]    2    4    6

is.array(a) #aは配列
#[1] TRUE

a[2,1] #2行1列目の要素
#[1] 2
a[2,3] #2行3列目の要素
#[1] 6


#コマンドの複製（replicate）

a = replicate(3, 1+1) #1+1の結果を3回繰り返しベクトルへ
#[1] 2 2 2
a = replicate(10,sample(1:6,1)) #サイコロの無作為抽出を10回繰り返す
#[1] 4 3 4 6 3 3 5 2 2 3
is.array(a) #aは配列ではなく、単なる（一次元）ベクトル
#[1] FALSE

#要素数n（>1）のベクトルを返す関数をm回replicateすると、
#結果はn x mの配列となる
a = replicate(10,sample(1:6,size=2))
#[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#[1,]    2    5    4    2    2    1    6    3    1     6
#[2,]    1    1    6    6    1    2    2    1    2     2
is.array(a) #aは配列
#[1] TRUE




#########################################
## （６）繰り返し文・条件文 ############
#########################################

#1:6に含まれる要素それぞれをaとし,
#aを出力する（print関数）
for(a in 1:6){
  print(a)
}
#[1] 1
```

```
#[1] 2
#[1] 3
#[1] 4
#[1] 5
#[1] 6


#for(a in vec)のaは必ずしも使う必要はない
#以下はサイコロの無作為抽出を5回繰り返したものの加算を計算したもの
sum = 0
for(a in 1:5){
  sum = sum + sample(1:6,1) #+=の記法はエラー（なぜ？）
  print(sum)
}
#[1] 2
#[1] 7
#[1] 10
#[1] 15
#[1] 18

#サイコロの総和が50以上となるまで加算
sum=0
while(sum<50){
  sum = sum + sample(1:6,1)
  print(sum)
}
#[1] 3
#[1] 5
#[1] 7
#中略
#[1] 35
#[1] 41
#[1] 45
#[1] 50

#repeat関数はwhile(true)に相当
#breakが実行されるまで永遠に実行する

sum = 0
repeat{
  sum = sum + sample(1:6,1)
  if(sum>50){
    break
  }
  print(sum)
}

#[1] 4
#[1] 10
#[1] 13
#中略
#[1] 41
#[1] 46
#[1] 49

##############################
## （７）独自関数 ############
##############################
```

```
#2個のサイコロを振って、総和を得るプログラム
dice = 1:6
dice2 = sample(dice, size=2, replace = TRUE)
sum(dice2)

#これをroll()として関数化する
roll = function(){
  dice = 1:6
  dice2 = sample(dice, size=2, replace = TRUE)
  sum(dice2)
}

#関数定義の最後の行が返り値となる（returnは使わないことに注意！！）
for(i in 1:3){
  ret = roll()
  print(ret)
}
#[1] 9
#[1] 10
#[1] 6


#n個のサイコロを振って、総和を得るプログラム
#引数ありの関数をつくります。
roll.n = function(n){
  dice = 1:6
  dice2 = sample(dice, size=n, replace = TRUE)
  sum(dice2)
}

result = vector("integer",10) #ベクトルサイズ10の中身ゼロの
for(i in 1:10){
  result[i] = roll.n(i)
}
#[1]  6  8 14 13 19 16 22 30 39 38




#roll関数を10000回繰り返し、一次元ベクトルに格納
rolls = replicate(1000, roll())
#[1]  6  7  9  6  9  4  3  9  8  4  7 10  8  8  7  9  8 10  7  8  8  6
#[23]  4 12  5  5 12  7 10  6  9  5  5  5 11  4 12  7  8 10  7 11  7  6
#[45]  5 12  8 11 10  7  5 11  9  5  9  7  5  4  8  6 10  9  5  6  9  9
#...
#[991]  6  6  7  7 10 10  5  9  9 12

#for文で書き直しています（同じことです）。
rolls=vector("integer") #integer型の空のベクトルを生成
for(i in 1:1000){
  rolls[i] = roll()
}

#あらかじめサイズを決める時は、以下のようにすると、
#全ての要素が0のサイズ1000のベクトルができます
rolls=vector(mode="integer",length=1000)
```

```
#ヒストグラムの計算（roll関数を使います）
imax = 10000
rolls = replicate(imax, roll())

hi=vector("integer",12) #中身が0の長さ12のベクトルをつくる
# [1] 0 0 0 0 0 0 0 0 0 0 0 0

for(i in 1:imax){
  hi[rolls[i]] = hist[rolls[i]]+1
}
# [1]   0  28  53  85  93 140 180 146 112  89  54  20
#棒グラフで可視化（barplot）
barplot(hi, main="Double dice",names.arg=1:12, ylab="Frequency")
#mainはタイトル
#names.argはx軸の値
#ylabはy軸のラベル

#hist関数を使うと、サンプルベクトルから自動でヒストグラムを生成できる
hist(rolls,breaks=max(rolls)-min(rolls),xlim=c(0,12),ylim=c(0,2000))
#breaksは分割数
```