

# 短期集中研修 『Rデータ解析自由自在（入門編）』

## ②データ・リテラシー

三村 喬生

統計数理研究所  
医療健康データ科学研究中心

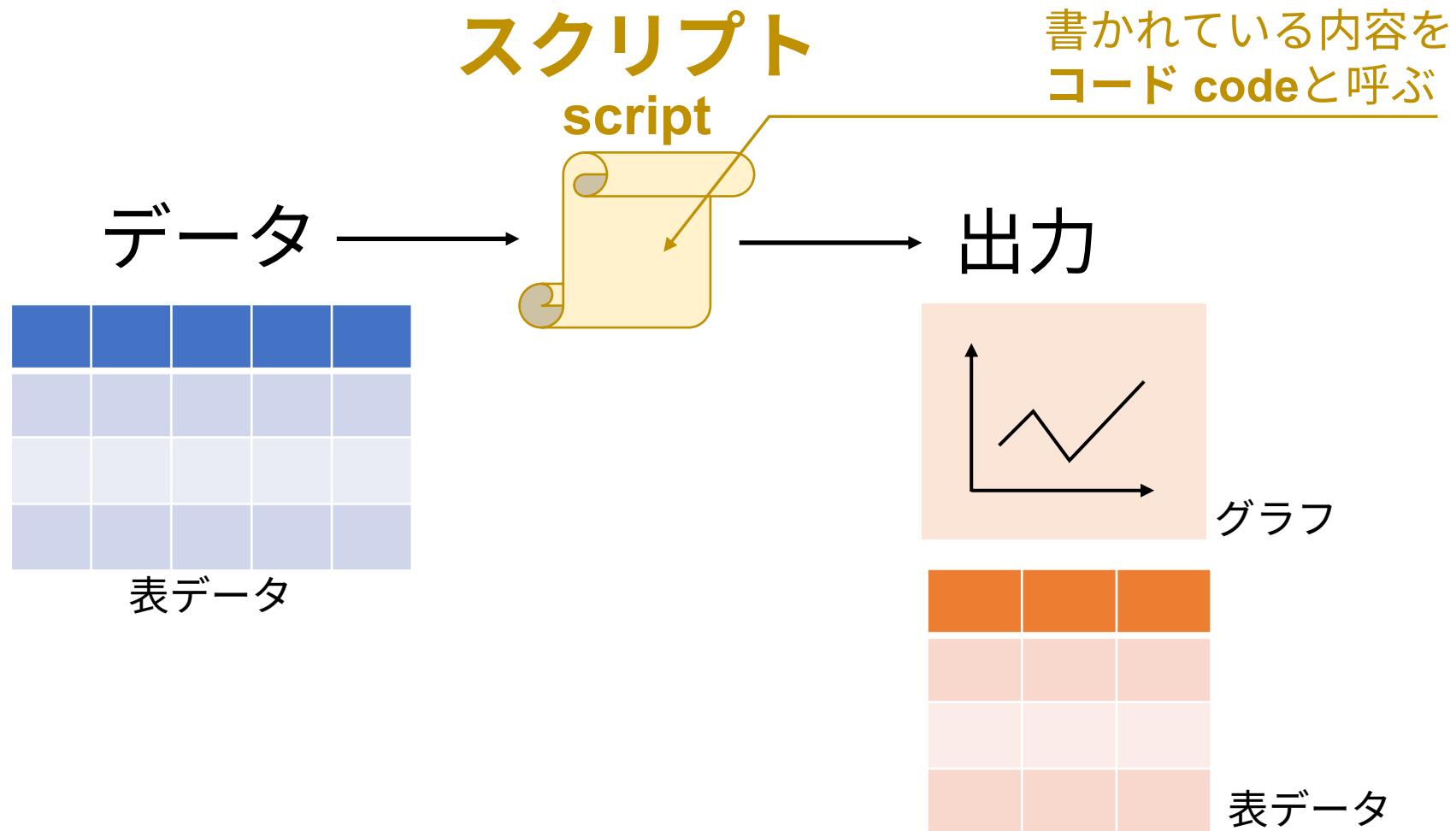
# プログラミング言語を使おう

## 基本的な姿勢

結果ではなく解析の工程を  
明示的に組み上げる

- ・自ずと結果も共有される。
- ・追加処理が容易になる。
- ・デバックしやすい。

# R = スクリプト言語



# R = スクリプト言語

## スクリプト

```
## packages ----
library(tidyverse)

## parameters ----
r <- 2
x_center <- 1
y_center <- 3

## data ----
df <-
tibble(
  theta = seq(0, 2 * pi, length = 361),
  X = r * cos(theta) + x_center,
  Y = r * sin(theta) + y_center
)

## visualization ----
ggplot(data = df) +
  aes(x = X, y = Y) +
  geom_path() +
  coord_fixed()
```

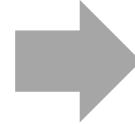
# ■ Rを始めよう

- ・基礎知識(オブジェクト, 関数)
- ・データの読み書き
- ・データの加工
- ・データの可視化

例えばExcelでは、

セルの内部に関数を書いて

a		5
x	y	
1	=B6*C\$3	
2		10
3		15
4		20
5		25



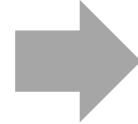
結果だけを表示する

a		5
x	y	
1		5
2		10
3		15
4		20
5		25

# 例えばExcelでは、

セルの内部に関数を書いて

a		5
x	y	
1	=B6*C\$3	
2		10
3		15
4		20
5		25



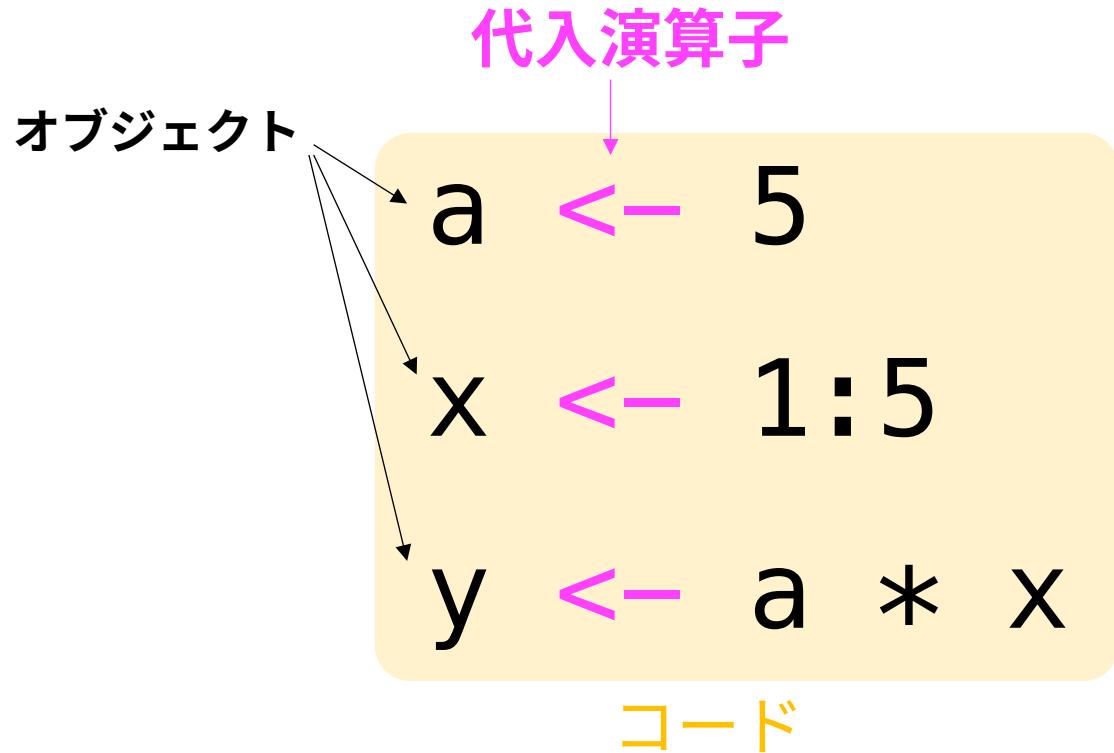
結果だけを表示する

a		5
x	y	
1		5
2		10
3		15
4		20
5		25

どこまでが共通の属性なのか  
直感的な配置から読み取る



# の基本



# ■ オブジェクト

- ・ベクトル vector
- ・リスト list
- ・データフレーム data.frame

# ■ ベクトル vector

Rでデータを格納する最小単位

```
x <- 1
```

```
> x  
[1] 1
```

# ■ ベクトル vector

ベクトル 代入演算子

`x <- 1`

```
> x  
[1] 1
```

# ■ ベクトル vector

文字型はダブルコーテーションで括る



```
x <- "あ"
```

```
> x  
[1] "あ"
```

# ■ ベクトル vector

複数の要素を持つベクトル

```
x <- c(1, 2, 3)
```

```
> x  
[1] 1 2 3
```

# ■ ベクトル vector

c()関数：括弧の中身を1つのベクトルに格納

```
x <- c(1, 2, 3)
```

```
> x  
[1] 1 2 3
```

# ■ ベクトル vector

1つベクトルの中身は1つの型

```
x <- c(1, 2, 3, "a")
```

```
> x  
[1] "1" "2" "3" "a"  
     ↑
```

文字型が混ざると他の数値型の要素も全て強制的に文字型の扱いになる。

# ■ ベクトル vector

要素へのアクセスは[ ]を使う

```
x <- c(1, 2, 3)
```

```
> x[1]  
[1] 1  
  
> x[c(2, 3)]  
[1] 2 3  
  
> x[c(3, 1)]  
[1] 3 1
```

# ■ ベクトル vector

seq()関数：規則的なベクトルを作る

```
x <- seq(from = 0, to = 8, by = 2)
```

```
> x  
[1] 0 2 4 6 8
```

# ■ ベクトル vector

```
x <- seq(from = 0, to = 8, by = 2)
```

引数                  引数                  引数  
↓                  ↓                  ↓  
from = 0, to = 8, by = 2

```
> x  
[1] 0 2 4 6 8
```

# ■ ベクトル vector

```
x <- seq(from = 0, to = 5,  
length.out = 3)
```

```
> x  
[1] 0 2.5 5.0
```

# ■ ベクトル vector

> ?seq

seq {base}

R Documentation

## Sequence Generation

### Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

### Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
```

# ■ ベクトル vector

```
> ?seq
```

## Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
```

Defaultの並び方に従っている限り引数名を省略できる。  
(引数名を省略するとDefaultで指定した順番だと解釈される)

# ■ ベクトル vector

```
x <- seq(from = 0, to = 8, by = 2)
```

```
x <- seq(0, 8, 2)
```

あまり省略に頼ると可読性が落ちる。  
byぐらいは書いたほうが親切かもしれない

# ■ ベクトル vector

```
x <- seq(8, 0, by = -2)
```

```
> x  
[1] 8 6 4 2 0
```

公差が負の値でもOK

# ■ ベクトル vector

```
x <- seq(8, 0, length.out = 10)
```

```
> x
[1] 8.0000000 7.5789474 7.1578947 6.7368421 6.3157895
[6] 5.8947368 5.4736842 5.0526316 4.6315789 4.2105263
[11] 3.7894737 3.3684211 2.9473684 2.5263158 2.1052632
[16] 1.6842105 1.2631579 0.8421053 0.4210526 0.0000000
```

割り切れなくてもOK

# ■ ベクトル vector

```
x <- seq(8, 0, length.out = 10)
```

```
> x
[1] 8.0000000 7.5789474 7.1578947 6.7368421 6.3157895
[6] 5.8947368 5.4736842 5.0526316 4.6315789 4.2105263
[11] 3.7894737 3.3684211 2.9473684 2.5263158 2.1052632
[16] 1.6842105 1.2631579 0.8421053 0.4210526 0.0000000
```

引数名は完全な表記でなくても  
前方一致で優先的に解釈される。

※ 可読性が落ちない範囲に留めましょう

# ■ ベクトル vector

rep()関数：繰り返し要素のベクトルを作る



```
x <- rep(x = 0, times = 5)
```

```
> x  
[1] 0 0 0 0 0
```

# ■ ベクトル vector

引数にベクトルを取れる

```
x <- rep(x = c(1, 2), times = 5)
```

```
> x  
[1] 1 2 1 2 1 2 1 2 1 2
```

# ■ ベクトル vector

```
a <- c(1, 2)
x <- rep(x = a, times = 5)
```

```
> x
[1] 1 2 1 2 1 2 1 2 1 2
```

## スクリプト

```
## packages ----  
library(tidyverse)  
  
## parameters ----  
r <- 2  
x_center <- 1  
y_center <- 3  
  
## data ----  
df <-  
  tibble(  
    theta = seq(0, 2 * pi, length = 361),  
    X = r * cos(theta) + x_center,  
    Y = r * sin(theta) + y_center  
  )  
  
## visualization ----  
ggplot(data = df) +  
  aes(x = X, y = Y) +  
  geom_path() +  
  coord_fixed()
```

## スクリプト

```
## packages ----  
library(tidyverse)  
  
## parameters ----  
r <- 2  
x_center <- 1  
y_center <- 3  
  
## data ----  
df <-  
  tibble(  
    theta = seq(0, 2 * pi, length = 361),  
    X = r * cos(theta) + x_center,  
    Y = r * sin(theta) + y_center  
)  
  
## visualization ----  
ggplot(data = df) +  
  aes(x = X, y = Y) +  
  geom_path() +  
  coord_fixed()
```

この部分

↓

↓

# ■ ベクトル vector

```
x <- seq(0, 8, length = 8)
```

```
> x
[1] 0.000000 1.142857 2.285714
[4] 3.428571 4.571429 5.714286
[7] 6.857143 8.000000
```

# ■ ベクトル vector

```
x <- seq(0, 8, length = 8 + 1)
```

```
> x  
[1] 0 1 2 3 4 5 6 7 8
```

## スクリプト

```
## packages ----
library(tidyverse)

## parameters ----
r <- 2
x_center <- 1
y_center <- 3

## data ----
df <-
  tibble(    ↓
    theta = seq(0, 2 * pi, length = 361),  ↓
    X = r * cos(theta) + x_center,
    Y = r * sin(theta) + y_center
  )

## visualization ----
ggplot(data = df) +
  aes(x = X, y = Y) +
  geom_path() +
  coord_fixed()
```

やってみよう

このコードを改変し、  
正n角形を描画するには！？

## スクリプト

```
## packages ----
library(tidyverse)

## parameters ----
r <- 2
x_center <- 1
y_center <- 3
n <- 13   #>

## data ----
df <-
  tibble(
    theta = seq(0, 2 * pi, length = n+1),
    X = r * cos(theta) + x_center,
    Y = r * sin(theta) + y_center
  )

## visualization ----
ggplot(data = df) +
  aes(x = X, y = Y) +
  geom_path() +
  coord_fixed()
```

# ■ リスト list

複数のベクトルを1つにまとめる

```
x <- list(c(1, 2, 3), c(4, 5))
```

```
> x  
[[1]]  
[1] 1 2 3
```

```
[[2]]  
[1] 4 5
```

# ■ リスト list

```
x <- list(c(1, 2, 3), c(4, "a"))
```

```
> x  
[[1]]  
[1] 1 2 3
```

```
[[2]]  
[1] "4" "a"
```

※ 1つベクトルの中身は1つの型

# ■ リスト list

要素へのアクセスは[[ ]]を使う

```
x <- list(c(1, 2, 3), c(4, 5))
```

```
> x[[2]][1]  
[1] 4
```

# ■ リスト list

要素には名前をつけることが可能

```
x <- list(a = c(1, 2, 3),  
           b = c(4, 5))
```

```
> x  
$a  
[1] 1 2 3
```

```
$b  
[1] 4 5
```

# ■ リスト list

要素には名前をつけることが可能

```
x <- list(a = c(1, 2, 3),  
           b = c(4, 5))
```

```
> x$a  
[1] 1 2 3  
> x$b[c(2, 1)]  
[1] 5 4
```

この場合は\$で要素にアクセスする。

# ■ リスト list

オブジェクト型はclass()関数で確認

```
x <- list(a = c(1, 2, 3),  
           b = c(4, 5))
```

```
> class(x)  
[1] "list"  
> class(x$a)  
[1] "numeric"
```

# ■ リスト list

概要を表示するstr()関数

```
x <- list(a = c(1, 2, 3),  
          b = c(4, 5))
```

```
> str(x)  
List of 2  
 $ a: num [1:3] 1 2 3  
 $ b: num [1:2] 4 5
```

# ■ データフレーム data.frame

- ・ 同一要素数のベクトルからなる名前付きlist
- ・ listの要素が列方向に整列され、順序ごとに行番号が対応づけられる

# ■ データフレーム data.frame

data.frame()関数で作成する

```
x <-  
  data.frame(  
    A = c(1, 2, 3),  
    B = c("a", "b", "c")  
)
```

```
> X  
  A B  
1 1 a  
2 2 b  
3 3 c
```

# ■ データフレーム data.frame

data.frame()関数で作成する

```
x <-  
  data.frame(  
    A = c(1, 2, 3),  
    B = c("a", "b", "c")  
)
```

```
> x  
  A B  
1 1 a  
2 2 b  
3 3 c
```

```
> is.list(x)  
[1] TRUE  
> is.data.frame(x)  
[1] TRUE
```

# ■ データフレーム data.frame

str()関数で概要を表示する

```
x <-  
  data.frame(  
    A = c(1, 2, 3),  
    B = c("a", "b", "c")  
)
```

```
> x  
   A B  
1 1 a  
2 2 b  
3 3 c
```

```
> str(x)  
'data.frame': 3 obs. of 2 variables:  
 $ a: num 1 2 3  
 $ b: chr "a" "b" "c"
```

# ■ データフレーム data.frame

名前付きリストと同様にアクセス

```
x <-  
  data.frame(  
    A = c(1, 2, 3),  
    B = c("a", "b", "c")  
)
```

```
> x  
  A B  
1 1 a  
2 2 b  
3 3 c
```

```
> x$B[c(3, 1)]  
[1] c a
```

# ■ データフレーム data.frame

オブジェクト名[行番号, 列番号]でもアクセスできる

```
x <-  
  data.frame(  
    A = c(1, 2, 3),  
    B = c("a", "b", "c")  
)
```

```
> x  
  A B  
1 1 a  
2 2 b  
3 3 c
```

```
> x [1, 2]  
[1] "a"
```

覚えておいて損はないけれど、行列と混乱するのであまり推奨しない

# ■ 行列 matrix

1つのベクトルを折りたたんだもの

```
a <- seq(1, 9)
x <- matrix(a, nrow = 3)
```

```
> x
      [,1] [,2] [,3]
[1,]     1     4     7
[2,]     2     5     8
[3,]     3     6     9
```

# ■ 行列 matrix

1つのベクトルを折りたたんだもの

```
a <- seq(1, 9)
x <- matrix(a, 3, byrow = TRUE)
```

```
> x
      [,1] [,2] [,3]
[1,]     1     2     3
[2,]     4     5     6
[3,]     7     8     9
```

# ■ 行列 matrix

1つのベクトルを折りたたんだもの

```
a <- seq(1, 9)
x <- matrix(a, 3, byrow = TRUE)
```

```
> x
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
> x[1, 2]
[1] 4
```

[行番号, 列番号]で  
アクセスする

# ■ 行列 matrix

転置行列はt()関数で求める

```
a <- seq(1, 9)
x <- matrix(a, 3, byrow = TRUE)
```

```
> x
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
> t(x)
 [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

# ■ 行列 matrix

行列の積は%\*%演算子で求める

```
a <- seq(1, 9)
x <- matrix(a, 3, byrow = TRUE)
y <- seq(1, 3)
```

```
> x %*% y
      [,1]
[1,]    14
[2,]    32
[3,]    50
```

# ■ データフレーム data.frame

- ・ 同一要素数のベクトルからなる名前付きlist
- ・ listの要素が列方向に整列され、順序ごとに行番号が対応づけられる
- ・ 1つのベクトルを折りたたんだ行列とは異なる

## スクリプト

```
## packages ----
library(tidyverse)

## parameters ----
r <- 2
x_center <- 1
y_center <- 3

## data ----
df <-
  tibble( tibble?
    theta = seq(0, 2 * pi, length = 361),
    X = r * cos(theta) + x_center,
    Y = r * sin(theta) + y_center
  )

## visualization ----
ggplot(data = df) +
  aes(x = X, y = Y) +
  geom_path() +
  coord_fixed()
```

```
> df <-  
+   data.frame(  
+     theta = seq(0, 2 * pi, length = 361),  
+     X = r * cos(theta) + x_center,  
+     Y = r * sin(theta) + y_center  
+   )
```

Error: object 'theta' not found

```
> df <-  
+   data.frame(  
+     theta = seq(0, 2 * pi, length = 361),  
+     X = r * cos(theta) + x_center,  
+     Y = r * sin(theta) + y_center  
+   )
```

Error: object 'theta' not found

オブジェクトdfを作り終わっていない段階では、  
theta列が作成されていないので、XやYを計算しようと  
思ってもtheta列が見つからないと怒られてしまう。

```
df <-  
  data.frame(  
    theta = seq(0, 2 * pi, length = 361)  
  ) %>%  
  mutate(  
    X = r * cos(theta) + x_center,  
    Y = r * sin(theta) + y_center  
  )
```

いったんtheta列だけのdata.frameを作つておいて、  
そこに後からX列やY列を追加で作成することは可能。  
(%>%演算子とmutate()関数については後ほど説明します)

# ■ ティブル tibble

- ・色々厳密になっているdata.frame
- ・遅延評価に対応(↓)

```
> df <-  
+   tibble(  
+     theta = seq(0, 2 * pi, length = 361),  
+     X = r * cos(theta) + x_center,  
+     Y = r * sin(theta) + y_center  
+   )
```

X列の作成指令を、先にtheta列を作り終わるまで保留するからエラーにならない。

# ■ ティブル tibble

- data.frameの拡張版
- 遅延評価に対応
- リストを列に取れる（後述）
- 行名は指定できない（数字のみ）
- tibbleパッケージを読み込むと  
利用できるようになる

# ■ ティブル tibble

Rをリスタート(⌘↑0)すると

```
> tibble
```

```
Error: object 'tibble' not found
```

# ■ ティブル tibble

tibbleパッケージを読み込んでおくと

```
> library(tibble)
> tibble
function (... , .rows = NULL, .name_repair =
  c("check_unique", "unique", "universal", "minimal"))
{
  xs <- quo(... )
  tibble_quos(xs, .rows, .name_repair)
}
<bytecode: 0x10df1d3f8>
<environment: namespace:tibble>
```

# ■ ティブル tibble

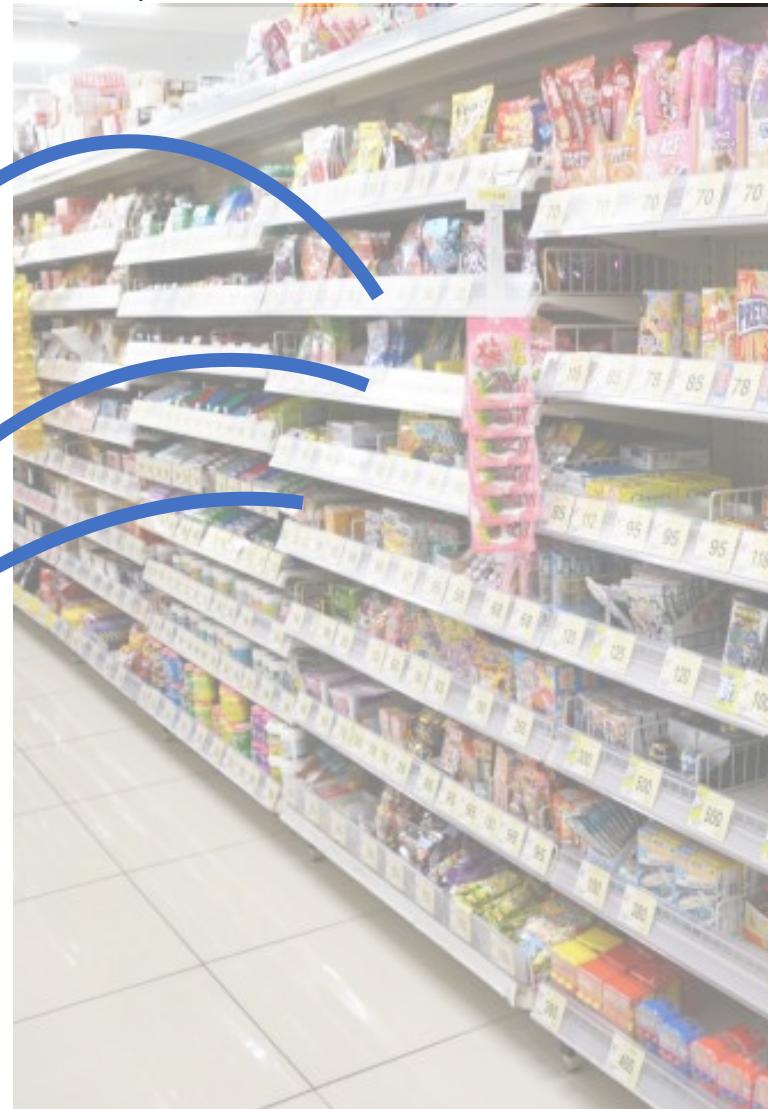
Rをリスタート(⌘↑0)において  
パッケージ名::関数名 でも認識される

```
> tibble::tibble
function (..., .rows = NULL, .name_repair =
  c("check_unique", "unique", "universal", "minimal"))
{
  xs <- quo(...)
  tibble_quos(xs, .rows, .name_repair)
}
<bytecode: 0x111099270>
<environment: namespace:tibble>
```

# パッケージ packages



CRAN  
The Comprehensive R Archive Network



# パッケージ packages

1. パッケージのインストール

```
install.packages("tidyverse")
```

2. インストールされたパッケージを読み込む

```
library(tidyverse)
```

# パッケージ packages

## Contributed Packages

### Available Packages

Currently, the CRAN package repository features **20115** available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

自分が思いついた事の99.9%は、既に世界中の誰かが  
思いついていて、何らかの足跡を残している。

→ 巨人の肩にドンドン乗りましょう。

# パッケージ packages

## CRAN R Packages by Number of Downloads

UPDATED DAILY. Last updated: 2023-11-30 01:17:20 +1100 Melbourne time (about 14 hours ago)

Rank	Package Name	Downloads	Author	Maintainer
1	<a href="#">ggplot2</a>	129,663,450		
2	<a href="#">rlang</a>	120,016,404		
3	<a href="#">magrittr</a>	116,774,666		
4	<a href="#">dplyr</a>	98,002,261		
5	<a href="#">vctrs</a>	85,651,791		
6	<a href="#">tibble</a>	82,867,055		
7	<a href="#">cli</a>	82,337,401		
8	<a href="#">devtools</a>	82,232,346		
9	<a href="#">jsonlite</a>	80,693,055		
10	<a href="#">pillar</a>	77,441,583		

## スクリプト

```
## packages ----  
library(tidyverse) ← tibbleパッケージは  
## parameters ----  
r <- 2  
x_center <- 1  
y_center <- 3  
## data ----  
df <-  
  tibble( ← なのにtibble()関数を  
    theta = seq(0, 2 * pi, length = 361),  
    X = r * cos(theta) + x_center,  
    Y = r * sin(theta) + y_center  
  )  
## visualization ----  
ggplot(data = df) +  
  aes(x = X, y = Y) +  
  geom_path() +  
  coord_fixed()
```

読み込んでない!?

なのにtibble()関数を  
使ってる?

# tidyverse



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

tidyverseパッケージはdplyr, ggplot2, stringrなど代表的なデータ科学関連パッケージをまとめて一括に共有するためのパッケージ。

# tidyverse

Rをリストアート(⌘↑0)しておいて

```
> sessionInfo()
```

R version 4.3.2 (2023-10-31)

Platform: aarch64-apple-darwin20 (64-bit)

Running under: macOS Ventura 13.5

...

attached base packages:

```
[1] stats   graphics grDevices  
[4] utils   datasets methods  
[7] base
```

loaded via a namespace (and not attached):

```
[1] utf8_1.2.4  
[2] R6_2.5.1  
[3] tidyselect_1.2.0  
[4] magrittr_2.0.3  
[5] glue_1.6.2  
[6] tibble_3.2.1  
[7] pkgconfig_2.0.3  
[8] dplyr_1.1.3
```

# tidyverse

```
> library(tidyverse)
```

```
> sessionInfo()
```

R version 4.3.2 (2023-10-31)

Platform: aarch64-apple-darwin20 (64-bit)

Running under: macOS Ventura 13.5

...

attached base packages:

```
[1] stats   graphics grDevices  
[4] utils   datasets methods  
[7] base
```

other attached packages:

```
[1] lubridate_1.9.3forcats_1.0.0 stringr_1.5.0  
[4] dplyr_1.1.3    purrr_1.0.2   readr_2.1.4  
[7] tidyr_1.3.0    tibble_3.2.1  ggplot2_3.4.4  
[10] tidyverse_2.0.0
```

loaded via a namespace (and not attached):

```
[1] utf8_1.2.4  
[2] R6_2.5.1
```

tidyverseにはこれ以外にも様々な  
パッケージが含まれ、必要に応じ  
て個別にlibrary()関数で読み込む



# tidyverse



{dplyr}: 表データの加工  
{ggplot2}: データ可視化  
{stringr}: 文字列の加工  
{magrittr}: 特殊演算子

R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

{purrr}: 置き込みデータ  
{tibble}: tibble型オブジェクト  
{readr}: データの読み書き  
{rmarkdown}: レポート作成

- オブジェクト
  - ・ベクトル vector
  - ・リスト list
  - ・データフレーム data.frame
  - ・行列 matrix
  - ・ティブル tibble

# ■ Rを始めよう

- ・基礎知識
- ・データの読み書き
- ・データの加工
- ・データの可視化

# ■ Rを始めよう

- ・基礎知識

- オブジェクトは名詞

- 関数は動詞

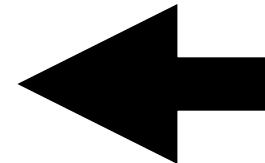
- ・データの読み書き

- ・データの加工

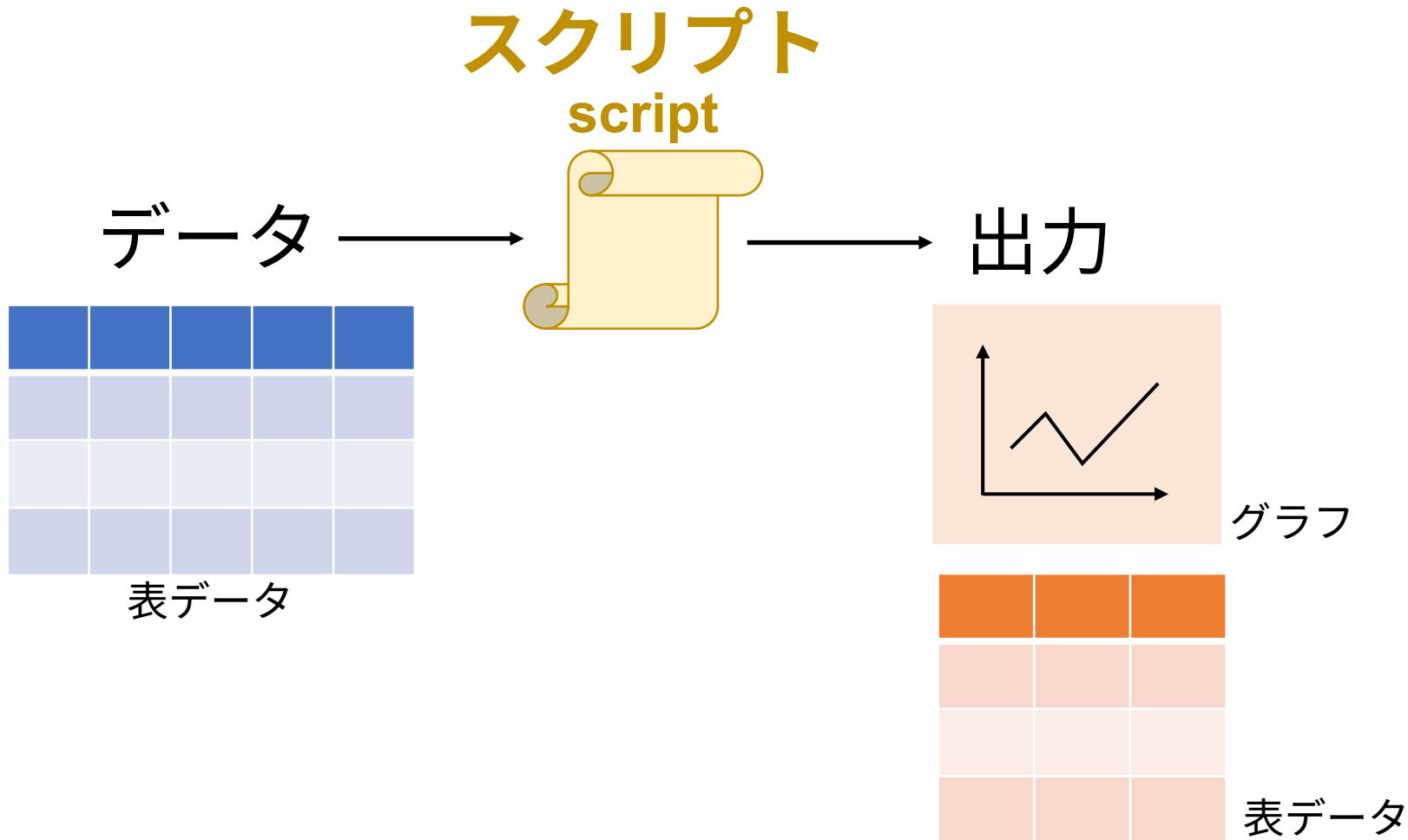
- ・データの可視化

# ■ Rを始めよう

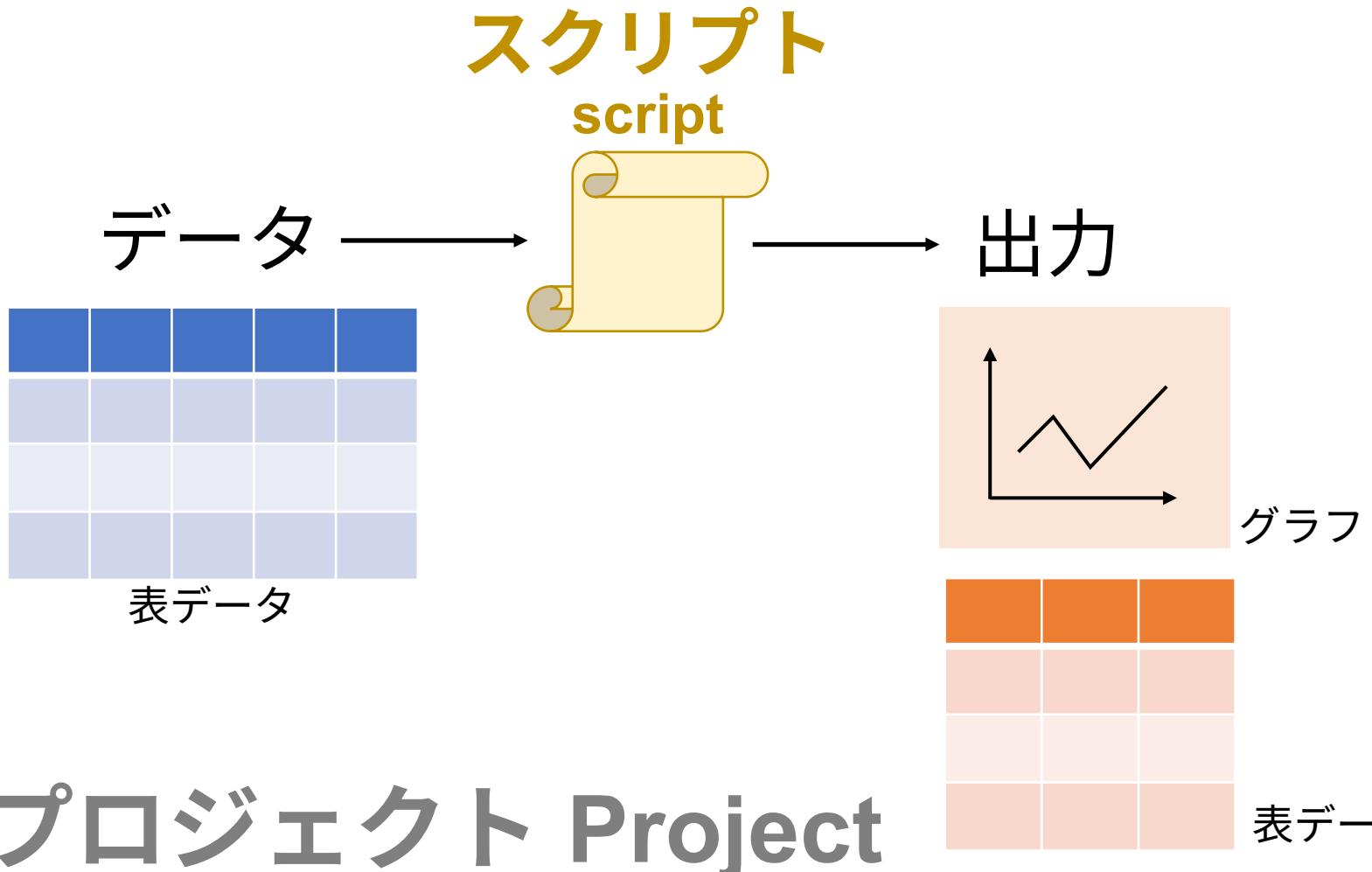
- ・基礎知識
  - オブジェクトは名詞
  - 関数は動詞
- ・データの読み書き
- ・データの加工
- ・データの可視化



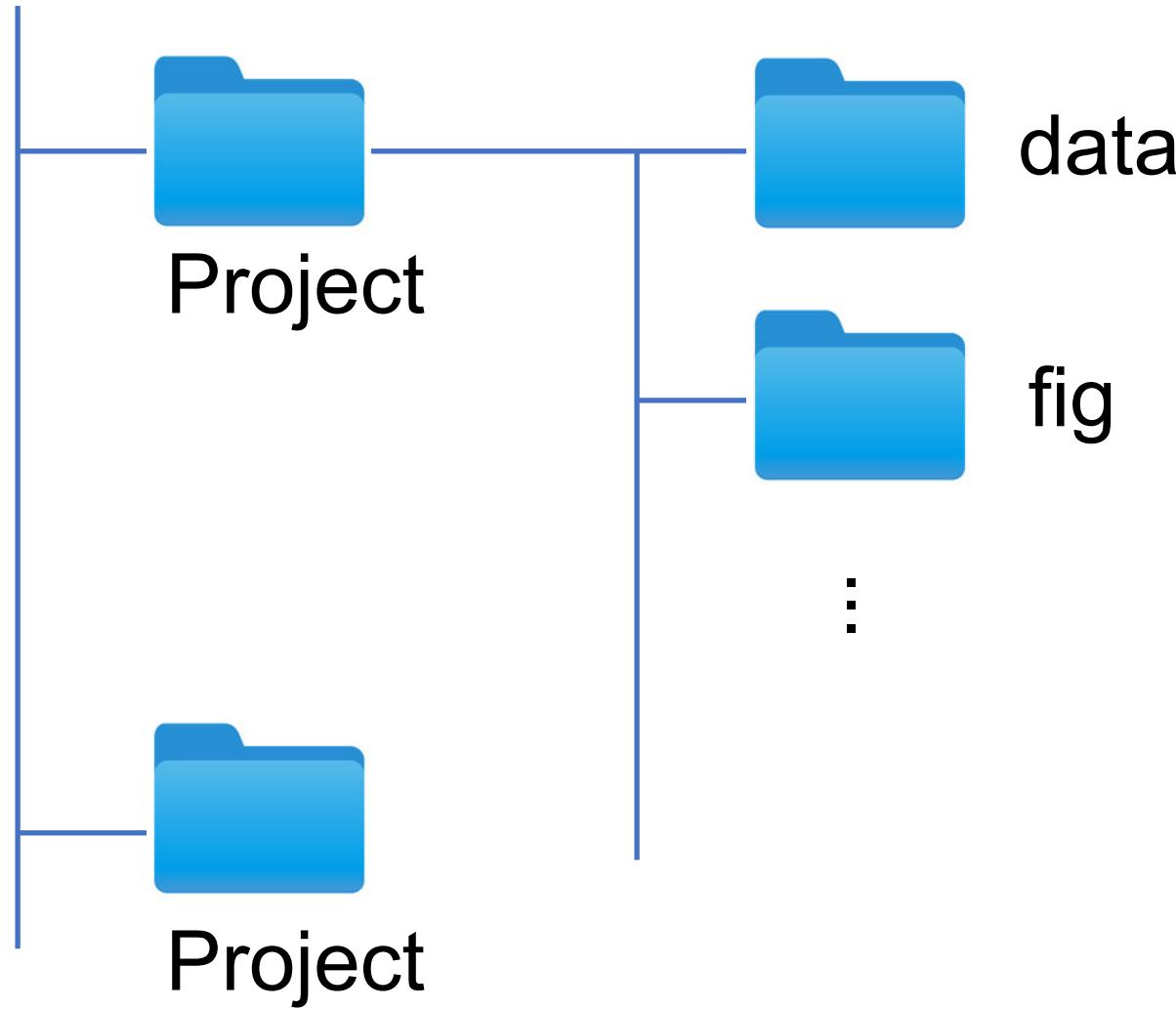
# ■ データの読み書き



# ■ データの読み書き



# ■ RStudioを使おう



# ■ RStudioを使おう

1. Rをインストール

<https://www.r-project.org/>

2. RStudioをインストール

<https://posit.co/download/rstudio-desktop/>

# ■ RStudioを使おう

3. RStudioを立ち上げる
4. 新規プロジェクトを作る  
File -> New Project...
5. 新規スクリプトを開く  
⌘ ↑ N

# RStudioの操作

The screenshot shows the RStudio interface with the following components:

- Top Bar:** Shows the title "tokyor107 - RStudio".
- Toolbar:** Includes icons for file operations like New, Open, Save, Print, and Go to file/function.
- Script Editor:** A dark-themed editor window titled "Untitled1" containing the text "スクリプト → 名前をつけて保存".
- Console:** Displays the R environment information and the message "Welcome to R".
- Output Area:** Shows the R command line history, including "demo()", "start()", and "type 'q()'". It also displays the message "\* Project ' ~/Documents/R/tokyor107' loaded. [renv 0.16.0]".
- Environment Tab:** Shows the "Environment" tab with the message "Environment is empty".
- File Explorer:** Shows the project structure under "Documents > R > tokyor107".

Large yellow annotations are overlaid on the screen:

- ① ここにRのコードを書く (written over the script editor)
- ② 選択して実行(⌘+⏎) (written over the console area, accompanied by a large yellow arrow pointing right)
- ③ 実行結果が表示 (written over the output area)

# RStudioの操作

The screenshot shows the RStudio interface with several Japanese annotations:

- スクリプト → 名前をつけて保存** (Script → Save with name) - An annotation pointing to the script editor tab.
- ① ここにRのコードを書く** (① Write R code here) - An annotation pointing to the script editor area.
- ② 選択して実行(⌘+⌃)** (② Run selected (⌘+⌃)) - An annotation pointing to the run button in the toolbar.
- ③ 実行結果が表示** (③ Execution results are displayed) - An annotation pointing to the console output area.
- 環境変数などが表示** (Environment variables are displayed) - An annotation pointing to the environment pane showing "Environment is empty".
- 自動で反映** (Automatically reflected) - An annotation pointing to the file browser pane.
- ファイル/プロット/ヘルプなど** (Files/Plots/Help etc.) - An annotation pointing to the file browser pane.

The RStudio interface includes:

- Top bar: Go to file/function, Addins, tokyor107
- Script Editor: Untitled1, Source on Save, Source
- Console: R 4.2.2, Welcome message, Natural language support, English locale
- File Browser: Home > Documents > R > tokyor107, showing files: .., .Rprofile, renv, renv.lock, tokyor107.Rproj
- Environment pane: Environment is empty
- History, Connections, Tutorial tabs
- Import Dataset, Global Environment buttons
- Viewer, Presentation tabs

# ■ RStudioを使おう

フォルダを作る

```
dir.create("data")
```

フォルダ/ファイルの存在を確認する

```
dir.exists("data")
```

フォルダ/ファイルを削除する

```
file.remove("data")
```

やってみよう

# data フォルダを作る

1. 現在の作業環境(ワークスペース, WS)の情報を取得  
ヒント : `getwd()` 関数を使う
2. WSに data フォルダが存在するか確認する  
ヒント : 文字列の結合には `paste()` 関数を使う
3. 存在していれば終了
4. 存在していないければ data というフォルダを作成する  
ヒント : 条件分岐は `if(条件){実行内容}`

# dataフォルダを作る

```
wd <- getwd()
path <- paste(wd, "/data", sep = "")

judge <- dir.exists(path)

if(judge == FALSE){
  dir.create(path)
}
```

# data フォルダを作る

```
wd <- getwd()
path <- paste0(wd, "/data")

judge <- dir.exists(path)

if(judge == FALSE){
  dir.create(path)
}
```

# dataフォルダを作る

```
wd <- getwd()
path <-
  stringr::str_c(wd, "/data")

judge <- dir.exists(path)

if(judge == FALSE){
  dir.create(path)
}
```

# dataフォルダを作る

```
wd <- getwd()
path <- paste0(wd, "/data")

judge <- dir.exists(path)

if(judge == TRUE){
}else{
  dir.create(path)
}
```

# data フォルダを作る

```
wd <- getwd()
path <- paste0(wd, "/data")

judge <- dir.exists(path)

if(judge != TRUE){
  dir.create(path)
}
```

# data フォルダを作る

```
wd <- getwd()
path <- paste0(wd, "/data")

judge <- dir.exists(path)

if( ! judge){
  dir.create(path)
}
```

# data フォルダを作る

```
wd <- getwd()
path <- paste0(wd, "/data")

if( ! dir.exists(path) ){
  dir.create(path)
}
```

# data フォルダを作る関数を作る

1. 現在の作業環境(ワークスペース, WS)の情報を取得
2. WSに data フォルダが存在するか確認する
3. 存在していれば終了
4. 存在していなければ data というフォルダを作成
5. 1~4を実行する関数を作成する

ヒント：関数の定義は以下のようにする

関数名 <- function(引数){実行内容}

# dataフォルダを作る関数を作る

```
mkfolder <- function(fname) {  
  wd <- getwd()  
  path <- paste0(wd, "/", fname)  
  
  if(! dir.exists(path)){  
    dir.create(path)  
  }  
}
```

```
mkfolder(fname = data)
```

# ブール演算子 Boolean Algebra

# equal to

A  **$==$**  B

# not equal to

A  **$!=$**  B

# or

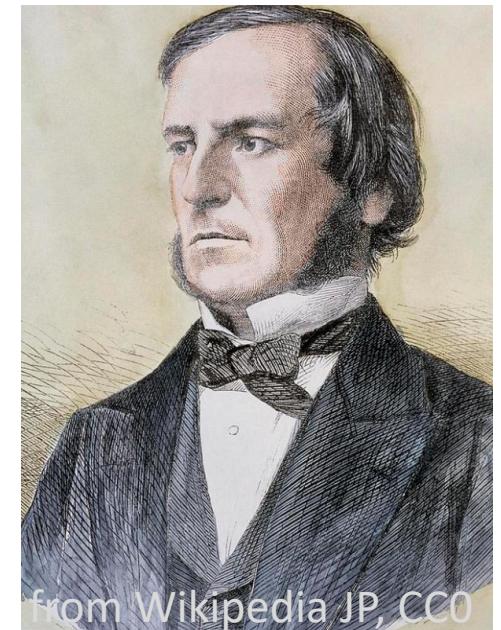
A  **$|$**  B

# and

A  **$\&$**  B

# is A in B?

A  **$\%in%$**  B

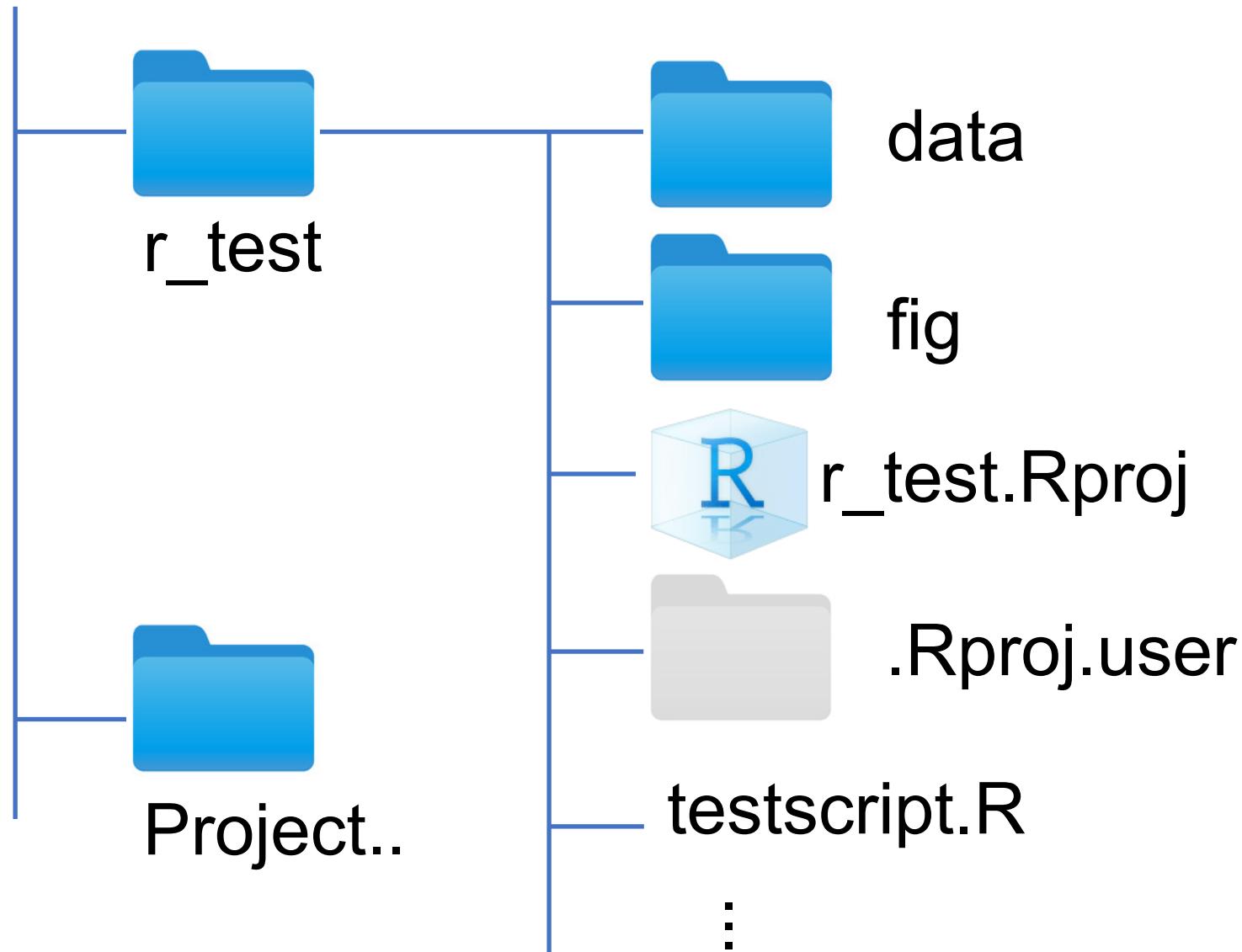


from Wikipedia JP, CC0

George Boole

1815 - 1864

# ■ RStudioを使おう



# ■ データを書き出す

> ChickWeight

Grouped Data: weight ~ Time | Chick

weight Time Chick Diet

1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1

# ■ データを書き出す

データの先頭を表示するhead()関数

head(ChickWeight)

Grouped Data: weight ~ Time | Chick  
weight Time Chick Diet

1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1

# ■ データを書き出す

データの先頭を表示するhead()関数

```
head(ChickWeight, n = 3)
```

```
Grouped Data: weight ~ Time | Chick  
weight Time Chick Diet
```

1	42	0	1	1
2	51	2	1	1
3	59	4	1	1

# ■ データを書き出す

データの末尾を表示するtail()関数

```
tail(ChickWeight, n = 3)
```

Grouped Data: weight ~ Time | Chick

	weight	Time	Chick	Diet
576	234	18	50	4
577	264	20	50	4
578	264	21	50	4

# ■ データを書き出す

変数ごとの数値の分布概要を表示するsummary()関数

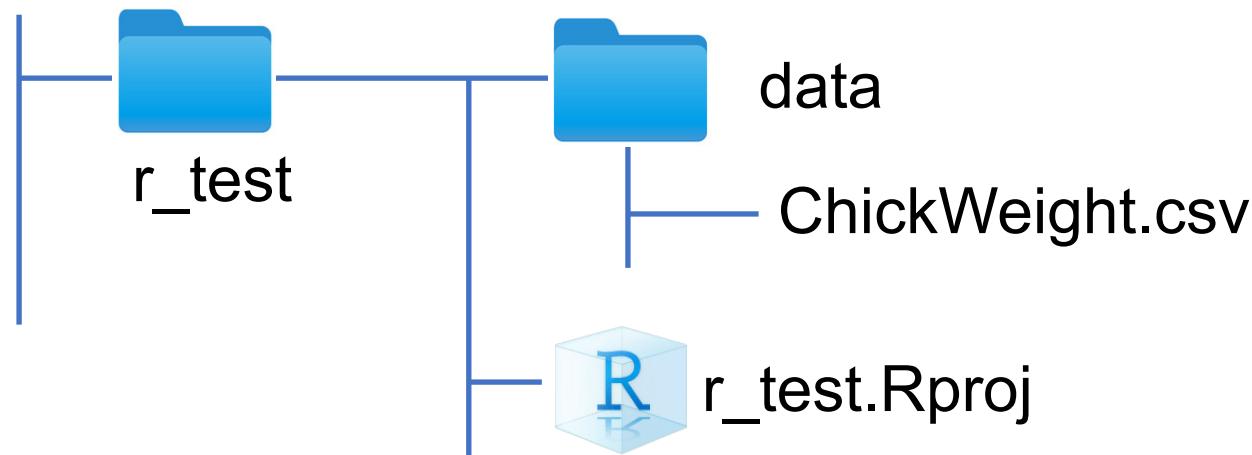
summary(ChickWeight)

weight	Time	Chick	Diet
Min. : 35.0	Min. : 0.00	13 : 12	1:220
1st Qu.: 63.0	1st Qu.: 4.00	9 : 12	2:120
Median :103.0	Median :10.00	20 : 12	3:120
Mean :121.8	Mean :10.72	10 : 12	4:118
3rd Qu.:163.8	3rd Qu.:16.00	17 : 12	
Max. :373.0	Max. :21.00	19 : 12	
		(Other):506	

# ■ データを書き出す

データを書き出す

```
write.csv(  
  x = ChickWeight,  
  file = "data/ChickWeight.csv"  
)
```



# ■ データを書き出す

行番号の列が勝手に入る

A	B	C	D	E
1	weight	Time	Chick	Diet
2	1	42	0	1
3	2	51	2	1
4	3	59	4	1
5	4	64	6	1
6	5	76	8	1
7	6	93	10	1
8	7	106	12	1
9	8	125	14	1
10	9	149	16	1
11	10	171	18	1
12	11	199	20	1
13	12	205	21	1

# ■ データを書き出す

データを書き出す

```
write.csv(  
  x = ChickWeight,  
  file = "data/ChickWeight.csv",  
  row.names = FALSE  
)
```

# ■ データを書き出す

データを書き出す

```
readr::write_csv(  
  x = ChickWeight,  
  file = "data/ChickWeight.csv"  
)
```

# ■ データを書き出す

データを書き出す

```
library(readr)

write_csv(
  x = ChickWeight,
  file = "data/ChickWeight.csv"
)
```

# ■ データを書き出す

データを書き出す

```
library(tidyverse)

write_csv(
  x = ChickWeight,
  file = "data/ChickWeight.csv"
)
```

# ■ データを読み込む

```
df <-  
  read.csv(  
    file = "data/ChickWeight.csv"  
)
```

# ■ データを読み込む

```
library(tidyverse)  
  
df <-  
  read_csv(  
    file = "data/ChickWeight.csv"  
)
```

read.csv()関数よりもread\_csv()関数の方が読み込みが早い。

# ■ データを読み込む

```
library(tidyverse)  
path <- "data/ChickWeight.csv"  
  
df <-  
  read_csv(file = path)
```

# ■ データを読み込む

```
library(tidyverse)  
path <- "data/ChickWeight.csv"  
  
df <-  
  read_csv(path)
```

# ■ データを読み込む

```
library(tidyverse)  
  
path <- "data/ChickWeight.csv"  
  
df <-  
  path %>%  
  read_csv()
```

# ■ データを読み込む

```
library(tidyverse)  
  
path <- "data/ChickWeight.csv"  
  
df <-  
  path %>%  
  read_csv()
```

↑ パイプ演算子

# パイプ演算子 pipe

直前の項を直後の関数の第一引数 (or 指定引数) に取る

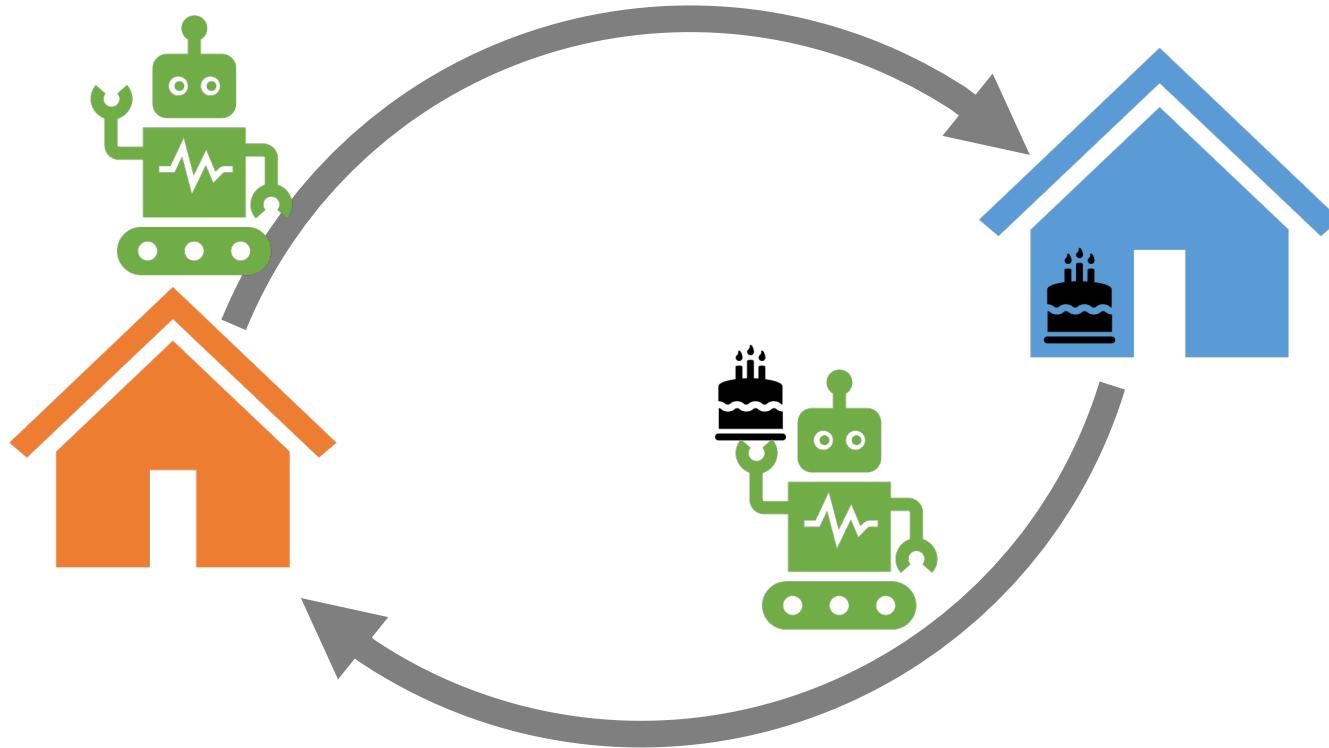
$$X \%>% f() \leftrightarrow f(X)$$

$$X \%>% f(y) \leftrightarrow f(X, y)$$

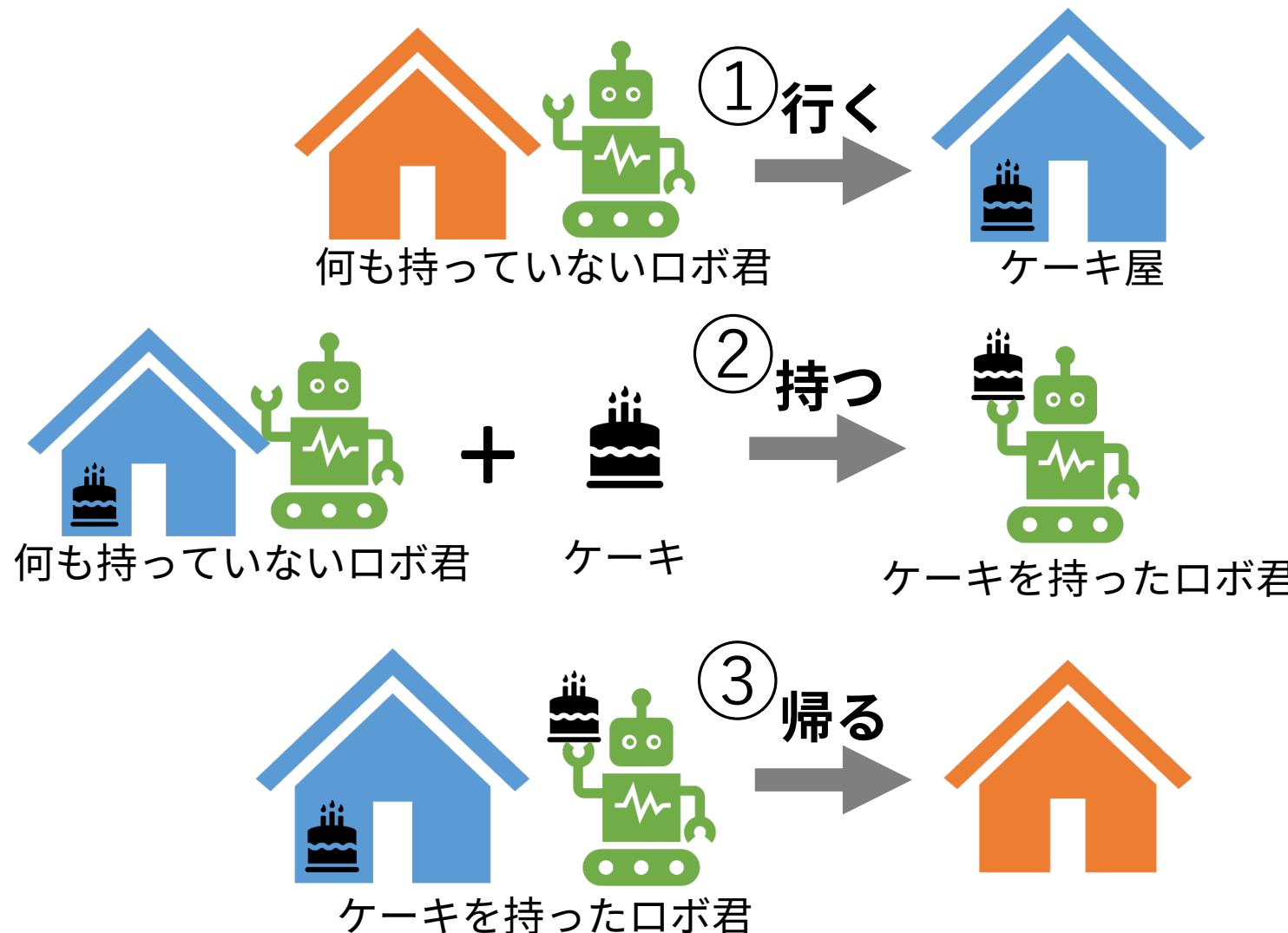
$$X \%>% f \%>% g \leftrightarrow g(f(X))$$

$$X \%>% f(y, .) \leftrightarrow f(y, X)$$

# 「口ボ君、ケーキを取ってきて」

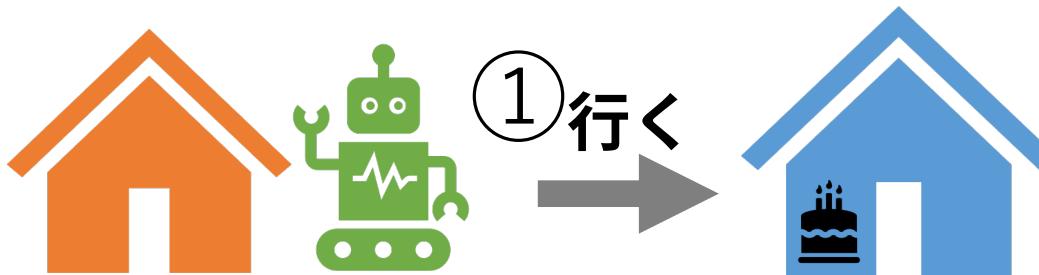


# 「口ボ君、ケーキを取ってきて」



3つの動詞 = 関数が含まれる

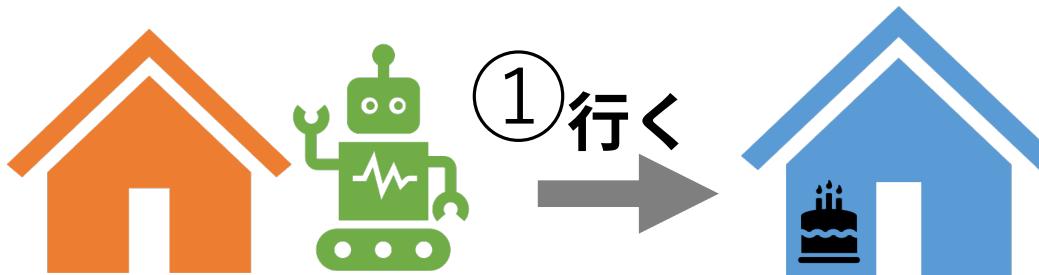
# 「口ボ君、ケーキを取ってきて」



① goto(口ボ君, from = 家, to = ケーキ屋)

3つの動詞=関数が含まれる

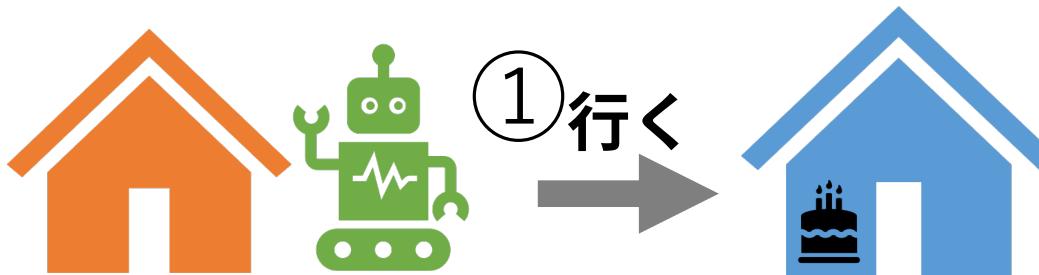
# 「口ボ君、ケーキを取ってきて」



- ① goto(口ボ君, from = 家, to = ケーキ屋)
- ② take(口ボ君, target = ケーキ)
- ③ goto(口ボ君, from = ケーキ屋, to = 家)

3つの動詞=関数が含まれる

# 「口ボ君、ケーキを取ってきて」



- ① goto(口ボ君, from = 家, to = ケーキ屋)  
家にいる暇な
- ② take(口ボ君, target = ケーキ)  
ケーキ屋に着いた
- ③ goto(口ボ君, from = ケーキ屋, to = 家)  
ケーキを持った

3つの動詞=関数が含まれる

# 「口ボ君、 ケーキを取ってきて」

- ① ケーキ屋に着いた口ボ君 <-  
    goto(家にいる暇な口ボ君,  
          from = 家, to = ケーキ屋)
- ② ケーキ屋でケーキを持った口ボ君 <-  
    take(ケーキ屋に着いた口ボ君,  
          target = ケーキ)
- ③ ケーキを取ってきた口ボ君 <-  
    goto(ケーキ屋でケーキを持った口ボ君,  
          from = ケーキ屋, to = 家)

3つの動詞=関数が含まれる

# 「口ボ君、 ケーキを取ってきて」

- ① 口ボ君b <- goto(口ボ君a, 家, ケーキ屋)
- ② 口ボ君c <- take(口ボ君b, target = ケーキ)
- ③ 口ボ君d <- goto(口ボ君c, ケーキ屋, 家)

3つの動詞=関数が含まれる

# 「口ボ君、 ケーキを取ってきて」

口ボ君b <- goto(口ボ君a, 家, ケーキ屋)

口ボ君c <- take(口ボ君b, target = ケーキ)

口ボ君d <- goto(口ボ君c, ケーキ屋, 家)



パイプを使う

口ボ君d <-

口ボ君a %>%

goto(from = 家, to = ケーキ屋) %>%

take(target = ケーキ) %>%

goto(from = ケーキ屋, to = 家)

# 「口ボ君、ケーキを取ってきて」

思考の流れ

口ボ君b <- goto(口ボ君a, 家, ケーキ屋)

口ボ君c <- take(口ボ君b, target = ケーキ)

口ボ君d <- goto(口ボ君c, ケーキ屋, 家)

読解の流れ

パイプを使う

口ボ君d <-

口ボ君a %>%

goto(from = 家, to = ケーキ屋) %>%

take(target = ケーキ) %>%

goto(from = ケーキ屋, to = 家)

# パイプ演算子 pipe

直前の項を直後の関数の第一引数 (or 指定引数) に取る

- 中間オブジェクトの乱立を回避できる
  - 中間オブジェクトの適切な名前を考えなくていい
  - オブジェクト(名詞)と処理(動詞)が明確になる
  - 環境を汚さない (地球に優しく!!)
- 見通しよくプログラムを組める
  - 思考とコードの流れ方を揃える事で全体の見通しup
  - 書きやすく読みやすいコードを書こう (ヒトに優しく!!)

# ■ データを読み込む

```
library(tidyverse)  
  
path <- "data/ChickWeight.csv"  
  
df <-  
  path %>%  
  read_csv()
```

↑ パイプ演算子

# ■ データを読み込む

**utils::read.csv()**

→ パッケージをインストールせずに使える  
読み込みが遅い

**readr::read\_csv()**

→ {tidyverse}に含まれる  
読み込みが速い, 多機能

**data.table::fread()**

→ 読み込みが超速い, 多機能

# ■ データを読み込む

`utils::read.csv()`

- パッケージをインストールせずに使える  
読み込みが遅い

`readr::read_csv()`

- `{tidyverse}`に含まれる  
読み込みが速い, 多機能  
複数ファイルをまとめて読み込む

`data.table::fread()`

- 読み込みが超速い, 多機能

やってみよう

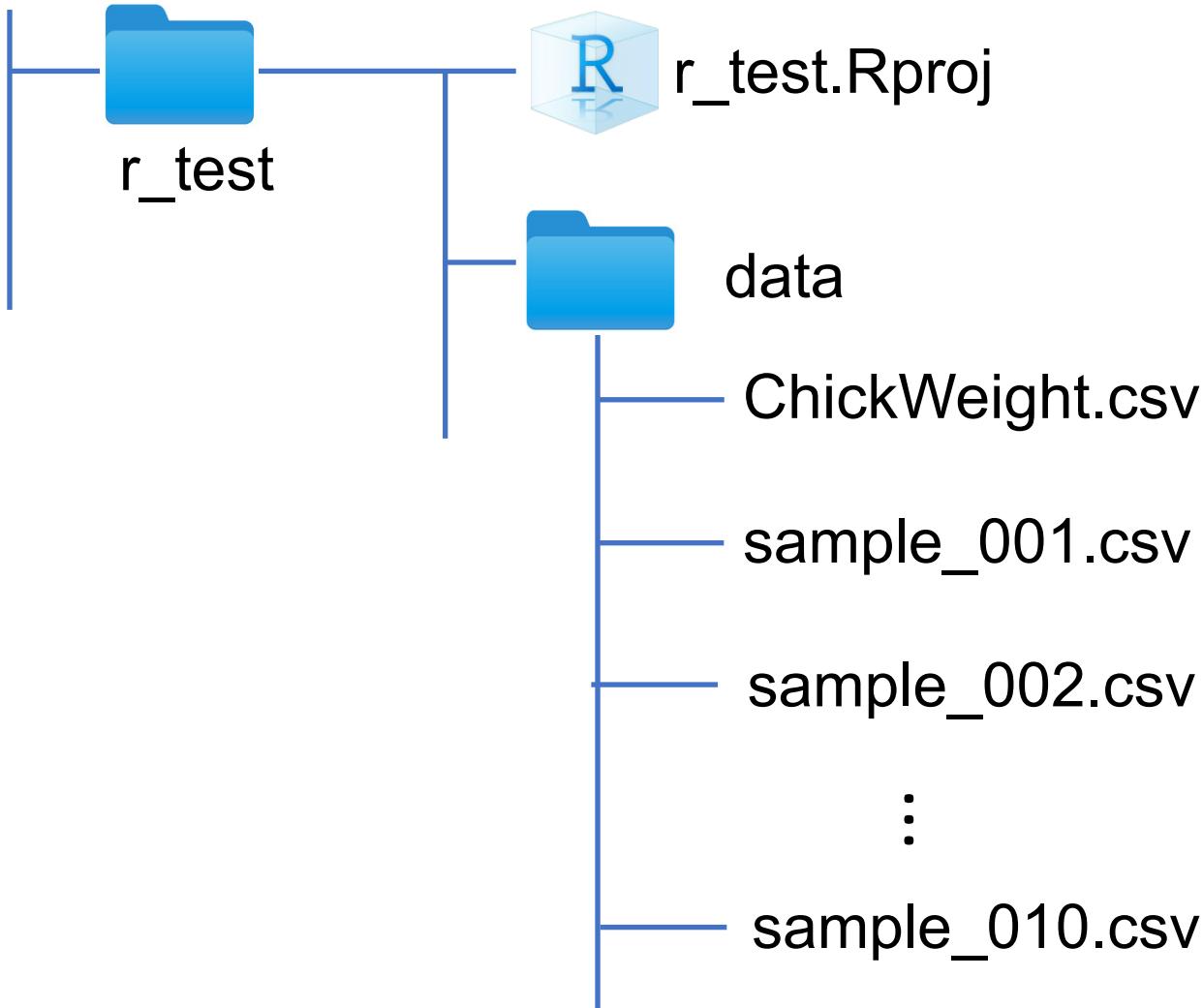
適当なdata.frameを作り、  
連番の名前を持ったcsvとして  
dataフォルダに10個出力しよう

やってみよう

適当なdata.frameを作り、  
連番の名前を持ったcsvとして  
dataフォルダに10個出力しよう

ヒント

- csvの出力はreadr::write\_csv()関数を使おう
- 連番ファイルの作成にはfor文を使おう
- 連番名の整形にはformatC()関数が便利



```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))
```

```
library(tidyverse)
```

```
path <- "data"
```

```
df <-
```

```
  data.frame(x = c(1, 2, 3),  
             y = c("a", "b", "c"))
```

```
for(i in 1:10){
```

ここで出力する

```
}
```

```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))
```

```
for(i in 1:10){
```

ここで出力先のパスpath\_outを作る

```
  df %>%
    write_csv(path_out)
}
```

```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))

for(i in 1:10){
  path_output <-
    paste0(path, "/sample_", i, ".csv")
    write_csv(df, path_out)
}
```

```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))

for(i in 1:10){
  path_output <-
    i %>%
    paste0(path, "/sample_", ., ".csv")

  df %>%
    write_csv(path_out)
}
```

```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))

for(i in 1:10){
  path_output <-
    i %>%
    formatC(width = 3, flag = "0") %>%
    paste0(path, "/sample_", ., ".csv")

  df %>%
    write_csv(path_out)
}
```

```
library(tidyverse)

path <- "data"

df <-
  data.frame(x = c(1, 2, 3),
             y = c("a", "b", "c"))

for(i in 1:10){
  path_output <-
    i %>%
    formatC(width = 3, flag = "0") %>%
    str_c(path, "/sample_", ., ".csv")

  df %>%
    write_csv(path_out)
}
```

やってみよう

作った10個のcsvファイルを  
まとめて1つのオブジェクトに  
読み込もう

ヒント

- `read_csv()`関数に読み込みたいファイルパスを  
渡せばOK。

```
library(tidyverse)

list_path <-
  seq(from = 1, to = 10, by = 1) %>%
  formatC(width = 3, flag = "0") %>%
  str_c("data/sample_", ., ".csv")
  ↑

df <-
  read_csv(list_path)
```

```
> df
# A tibble: 30 × 2
      x   y
  <dbl> <chr>
1     1   a
2     2   b
3     3   c
4     1   a
```

```
library(tidyverse)

list_path <-
  seq(from = 1, to = 10, by = 1) %>%
  formatC(width = 3, flag = "0") %>%
  str_c("data/sample_", ., ".csv")

df <-
  read_csv(list_path, id = "file_path")
```

```
> df
# A tibble: 30 × 3
  file_path          x y
  <chr>            <dbl> <chr>
1 data/sample_001.csv    1 a
2 data/sample_001.csv    2 b
3 data/sample_001.csv    3 c
4 data/sample_002.csv    1 a
5 data/sample_002.csv    2 b
6 data/sample_002.csv    3 c
7 data/sample_003.csv    1 a
8 data/sample_003.csv    2 b
9 data/sample_003.csv    3 c
10 data/sample_004.csv   1 a
# ... with 20 more rows
```

```
library(tidyverse)

list_path <-
  seq(from = 1, to = 10, by = 1) %>%
  formatC(width = 3, flag = "0") %>%
  str_c("data/sample_", ., ".csv")
```

```
df <-
  list_path %>%
  read_csv(id = "file_path")
```

```
> df
# A tibble: 30 × 3
  file_path          x y
  <chr>            <dbl> <chr>
1 data/sample_001.csv    1 a
2 data/sample_001.csv    2 b
3 data/sample_001.csv    3 c
4 data/sample_001.csv    1 a
5 data/sample_001.csv    2 b
6 data/sample_001.csv    3 c
7 data/sample_001.csv    1 a
8 data/sample_001.csv    2 b
9 data/sample_001.csv    3 c
10 data/sample_001.csv   10 a
# ... with 20 more rows
```

```
library(tidyverse)

list_path <-
  "data" %>%
  list.files(full.names = TRUE) %>%
  str_subset(pattern = "sample_")

df <-
  list_path %>%
  read_csv(id = "file_path")
```

```
library(tidyverse)

path_folder <- "data"
key <- "sample_"

list_path <-
  path_folder %>%
    list.files(full.names = TRUE) %>%
    str_subset(pattern = key)

df <-
  list_path %>%
  read_csv(id = "file_path")
```

```
## packages ----
library(tidyverse)

## parameters ----
path_folder <- "data"
key <- "df_"

## path ----
list_path <-
  path_folder %>%
  list.files(full.names = TRUE,
            pattern = key)

## import ----
df <-
  list_path %>%
  read_csv(id = "file_path")
```

```
## packages ----  
library(tidyverse)
```

よくできました！

```
## parameters ----  
path_folder <- "data"  
key <- "sample_"
```

```
## path ----  
list_path <-  
  path_folder %>%  
  list.files(full.names = TRUE) %>%  
  str_subset(pattern = key)
```

```
## import ----  
df <-  
  list_path %>%  
  read_csv(id = "file_path")
```

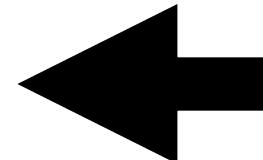
# ■ Rを始めよう

- 基礎知識

- オブジェクトは名詞

- 関数は動詞

- データの読み書き



- + プロジェクト管理 in RStudio

- + パッケージの利用

- + パイプ演算子

- + if文, for文, ディレクトリ操作など

# ■ Rを始めよう

- ・基礎知識(オブジェクト, 関数)
- ・データの読み書き
- ・データの加工
- ・データの可視化

*to be continued...*