心理統計の授業中に
GUIのwebアプリを作って遊ぼう

Shiny 入門②

日本心理学会 第85回大会

チュートリアル・ワークショップ

企画者 豊田秀樹・馬景昊

講師 豊田秀樹・馬景昊・堀田晃大

本チュートリアルの目的

- 私たちPCユーザーは日頃、
  統計処理をはじめ様々な命令をPCに伝えて作業させている

- ユーザーが見る画面の形式は2種類に大別される
  - CUI Character User Interface　コマンドを使って操作する
    例：R、プログラミング言語
  - GUI Graphic User Interface　ボタンやメニューを使って操作する
    例：SPSS、スマホのアプリ

チュートリアルの目的

- CUIは幅広い応用が可能な一方、最初に覚えるのが大変
- GUIは操作が簡単で初心者でも分かりやすい

- CUIであるRのパッケージShinyを利用して
  GUIで動かせるウェブアプリケーションを作る

チュートリアルの目的

ファイルと変数を指定してヒストグラムを描く

拡張子が csv か datのファイルを選択

参照 EuStockMarkets.csv

Upload complete

ヘッダーがあればチェック

区切り文字

カンマ
セミコロン
タブ

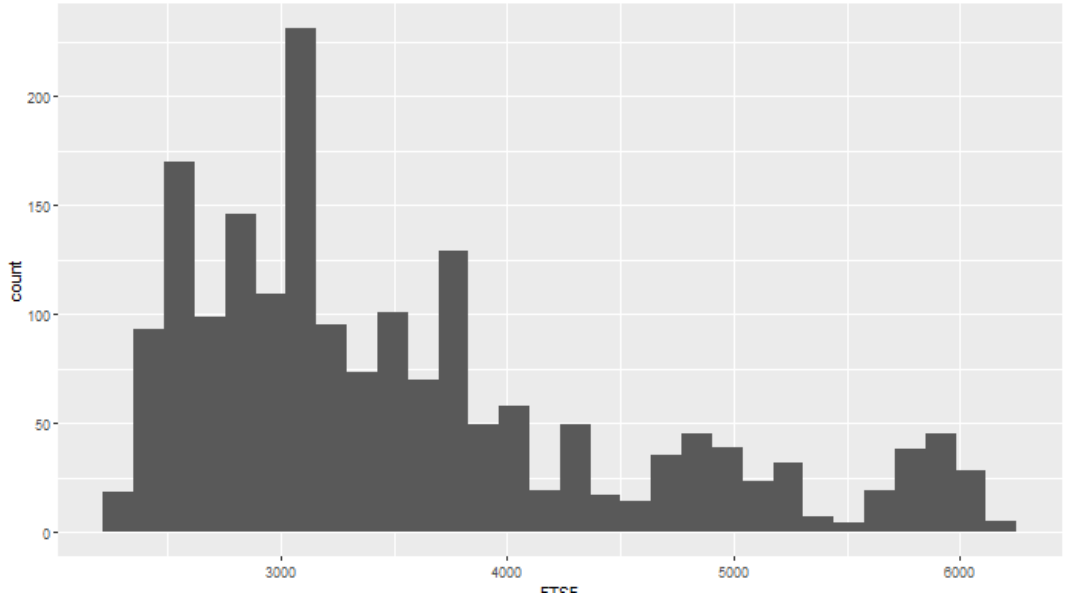
データセットの閲覧

最初
全部

変数を選んでください:

FTSE

DAX	SMI	CAC	FTSE
1628.75	1678.10	1772.80	2443.60
1613.63	1688.50	1750.50	2460.20
1606.51	1678.60	1718.00	2448.20
1621.04	1684.10	1708.10	2470.40
1618.16	1686.60	1723.10	2484.70
1610.61	1671.60	1714.30	2466.80



- 例えばこんなウェブアプリが自分で作れるようになる

本チュートリアルの流れ

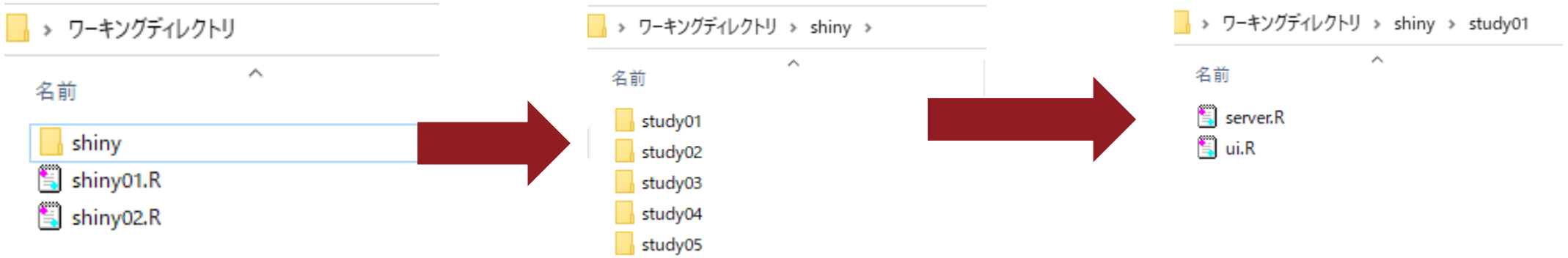
1. Shinyの基本構造
2. 簡単な例の実装
   （階級値の数を変更できるヒストグラム）
3. タグ関数　　（文章の表示）
4. レイアウト関数　（ボタンなどの操作可能なウィジェット）
5. 関数Reactive　　（快適な動作のために）
6. デプロイメント　（作成したアプリの公開方法）

本チュートリアルの流れ

1. Shinyの基本構造
2. 簡単な例の実装
   （階級値の数を変更できるヒストグラム）
3. タグ関数　　（文章の表示）
4. レイアウト関数　（ボタンなどの操作可能なウィジェット）
5. 関数Reactive　　（快適な動作のために）
6. デプロイメント　（作成したアプリの公開方法）

Shinyの基本構造

- Rのワーキングディレクトリの下に、
  以下のようなフォルダ構造を用意する



- Study01等のフォルダ1つ1つがアプリケーション
- 各フォルダの中にserver.Rとui.Rを格納
- 文字コードをUTF-8にすれば日本語表示できる

Shinyの基本構造

- Shiny01.Rにはアプリ起動用のスクリプトを記述
- 関数runAppでアプリを起動する

```
shiny::runApp('./shiny/study01')
shiny::runApp('./shiny/study02', display.mode = "showcase")
shiny::runApp('./shiny/study03')
```

- フォルダの相対パス（絶対パスでも良い）
- Display.mode＝“showcase” でスクリプトもアプリに表示

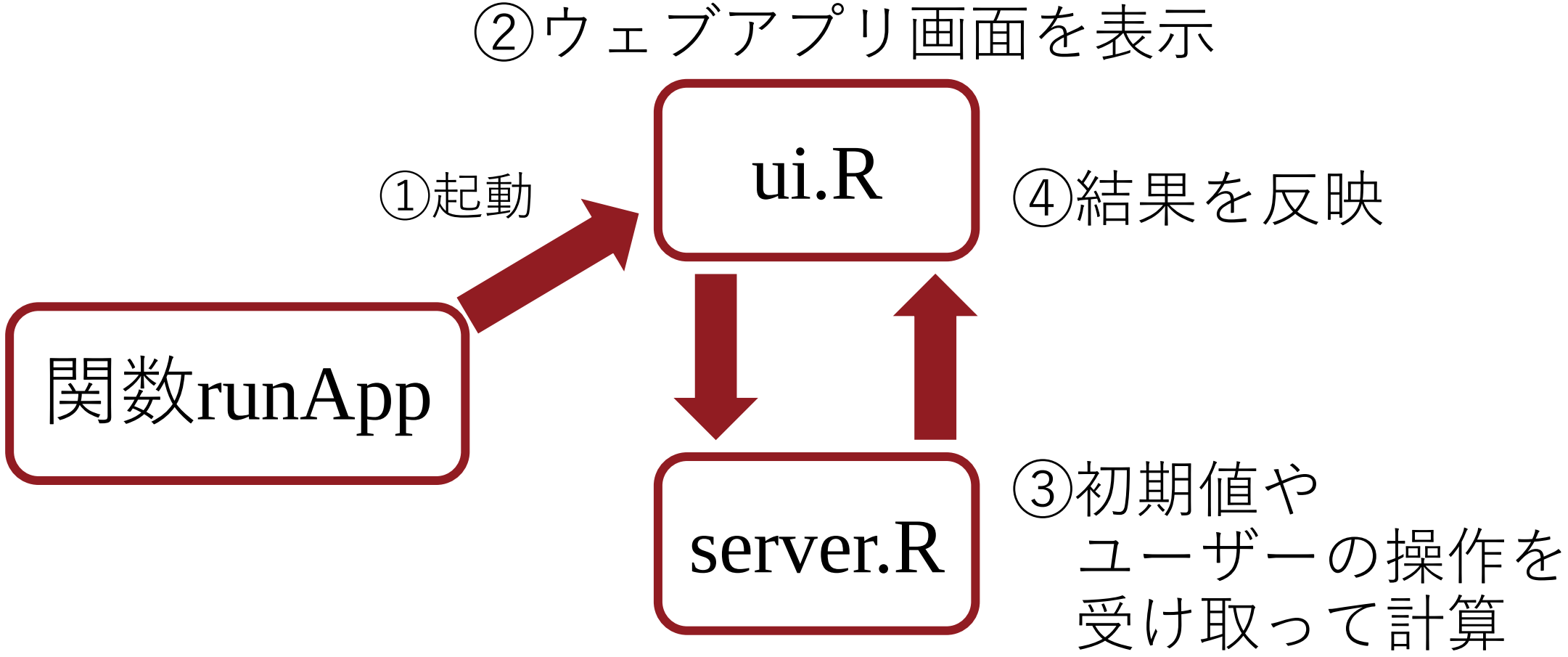
Shinyの基本構造

- ui.RにはUser Interface（画面に表示する部分）を記述
  関数shinyUIを用いる
- server.Rには分析ロジック（PCに計算させる部分）を記述
  関数shinyServerを用いる

```
#ui.R
↓
library(shiny)↓
shinyUI(fluidPage(↓
  # タイトル↓
  titlePanel("階級値の数を変更できるヒストグラム"),↓
  # レイアウトの設定↓
  sidebarLayout(↓
    sidebarPanel( "サイドバーパネルの指定"),↓
    mainPanel("メインパネルの指定")↓
  )↓
))↓
```

```
#server.R↵
↵
library(shiny)↵
shinyServer(function(input, output) {↵
})↵
↵
```

Shinyの基本構造


②ウェブアプリ画面を表示
①起動
ui.R
④結果を反映
関数runApp
server.R
③初期値や
ユーザーの操作を
受け取って計算

本チュートリアルの流れ

1. Shinyの基本構造
2. 簡単な例の実装
   （階級値の数を変更できるヒストグラム）
3. タグ関数　　（文章の表示）
4. レイアウト関数　（ボタンなどの操作可能なウィジェット）
5. 関数Reactive　　（快適な動作のために）
6. デプロイメント　（作成したアプリの公開方法）

簡単な例の実装

- 階級値の数を変更できる
  ヒストグラムを実装してみよう

- パッケージshinyを読み込む

- タイトル

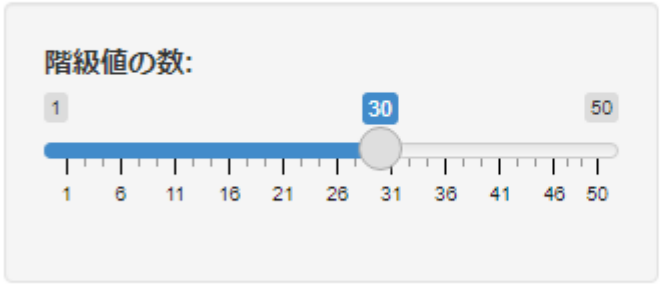
- sidebarLayoutの中で、
  サイドバーパネルと
  メインパネルに分岐

```
#ui.R
library(shiny)
shinyUI(fluidPage(

  # タイトル
  titlePanel("階級値の数を変更できるヒストグラム"),

  # レイアウトの設定
  sidebarLayout(
    # サイドバーパネルの指定
    sidebarPanel(
      # スライダーの設定
      sliderInput("bins",
                  "階級値の数:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    # メインパネルの指定
    mainPanel(
      plotOutput("distPlot")   # ヒストグラムの表示
    )
  )
))

```

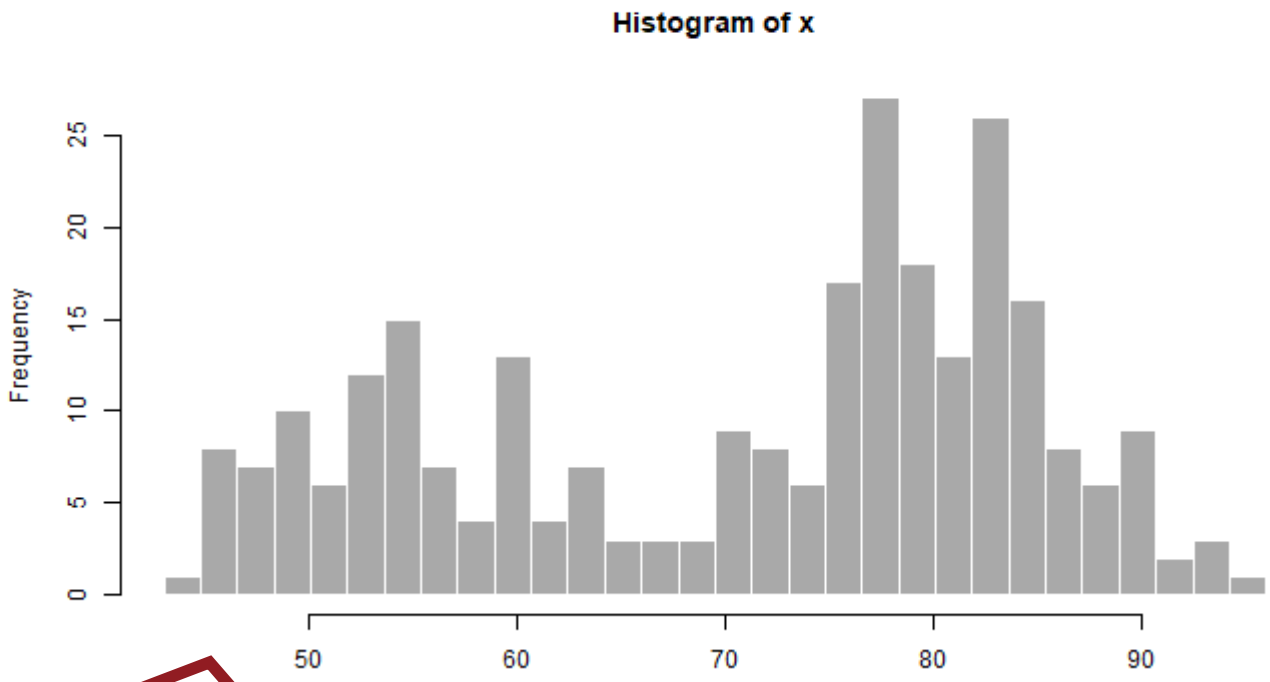
簡単な例の実装

タイトル

階級値の数を変更できるヒストグラム



サイドバーパネル



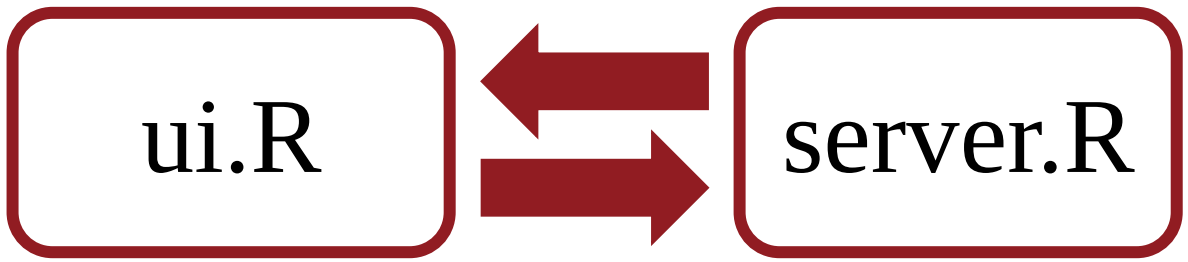
メインパネル

簡単な例の実装

```
#ui.R
library(shiny)
shinyUI(fluidPage(
  # タイトル
  titlePanel("階級値の数を変更できるヒストグラム"),
  # レイアウトの設定
  sidebarLayout(
    # サイドバーパネルの指定
    sidebarPanel(
      # スライダーの設定
      sliderInput("bins",
                  "階級値の数:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    # メインパネルの指定
    mainPanel(
      plotOutput("distPlot")  # ヒストグラムの表示
    )
  )
))
```

- 受け渡しする情報に名前をつける

```
#server.R
library(shiny)
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    # uiから受け取ったbins情報をもとに階級幅の生成
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    # ヒストグラムの生成
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```



簡単な例の実装

- Server.R内では
- input$*** で ui.R 側で入力された***を受け取る
- output$*** で 計算結果を *** という名前で ui.R側で取り出せるようにする

- ui.Rの関数 sliderInput (inputId, label, min, max, value)

```
# スライダーの設定
sliderInput("bins",
            "階級値の数:",
            min = 1,
            max = 50,
            value = 30)
```

引数

inputId server.R に渡す値

label スライダーのラベル

min 表示するスライダーの最小値

max 表示するスライダーの最大値

value server.R に渡す初期値

簡単な例の実装

- ui.Rのスライダーからの情報を”input$bins”で受け取る
- それを使って、server.R側の
  関数renderPlotでヒストグラムを生成

```
#server.R
↓
library(shiny)
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    # uiから受け取ったbins情報をもとに階級幅の生成
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    # ヒストグラムの生成
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
↓
```

- 完成したヒストグラムに”distPlot”と命名

簡単な例の実装

```
#ui.R
library(shiny)
shinyUI(fluidPage(
  # タイトル
  titlePanel("階級値の数を変更できるヒストグラム"),
  # レイアウトの設定
  sidebarLayout(
    # サイドバーパネルの指定
    sidebarPanel(
      # スライダーの設定
      sliderInput("bins",
                  "階級値の数:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    # メインパネルの指定
    mainPanel(
      plotOutput("distPlot")  # ヒストグラムの表示
    )
  )
))
```

- 関数plotOutputで”distPlot”を呼び出して描画

簡単な例の実装

- ここで、実際にアプリを起動してみましょう

本チュートリアルの流れ

1. Shinyの基本構造
2. 簡単な例の実装
   （階級値の数を変更できるヒストグラム）
3. タグ関数　　（文章の表示）
4. レイアウト関数　（ボタンなどの操作可能なウィジェット）
5. 関数Reactive　（快適な動作のために）
6. デプロイメント　（作成したアプリの公開方法）

タグ関数

- パネル内に文章を表示するにはタグ関数を用いる
- この関数は本質的にhtmlのタグと同等（cssも使える）

ui.R　mainPanel内　例

```
p("p は、テキストの段落を作成します。この関数の後は改行されます。"),
p("新しい段落を開始する場合は、新しい p() コマンドを使用します。"),
h1("h1:統計学入門", align = "center"),
h2("h2:回帰分析"),
h3("h3:偏回帰係数"),
h4("h4:応用例"),
h5("h5:注意点", align = "right", style = "color:green"),
h6("h6:予測変数が多い場合には、偏回帰係数を直接解釈してはいけない"),
p("コードは code()で以下のように表示します。"),
```

タグ関数

- htmlとはウェブページを記述する言語のこと

- 例えば、tags$head()　tags$ul()　tags$li()
  などと書けばhtmlのそれぞれ対応するタグが使える
- いくつかのよく使うタグはtags$を省略できる

- この節では、
  htmlを全く知らない方のための解説をおこなう

タグ関数

- タグには1つ1つ役割がある

- p　　テキストを1つの段落として表示
- p(“ここに書いた文章が1つの段落”)
- p(“次のpタグの中身は改行して2段落目”)

タグ関数

- p　　テキストを1つの段落として表示

- h1　　見出し
（h2からh6まで、順に文字サイズが小さくなる）
- code　　テキストをコードとして表示
- pre　　テキストをそのまま等幅フォントで表示

- img　　画像を表示
- br　　改行

タグ関数

- オプションとして、装飾ルール（style）を指定できる
- p(“文章”,　align=“cetner”,
  style=“color:blue,　font-size:16pt”)

- styleの中で使うのは、cssというhtml装飾用の言語

- 太字や斜字にするタグもある
- strong(“太字・ボールド体になる”)
- p(”段落の”,　em(“一部だけ”),　”斜字・イタリック体になる”)

タグ関数

- 一定の範囲をくくるタグ
- その範囲にまとめて装飾ルール（style）を適用できる

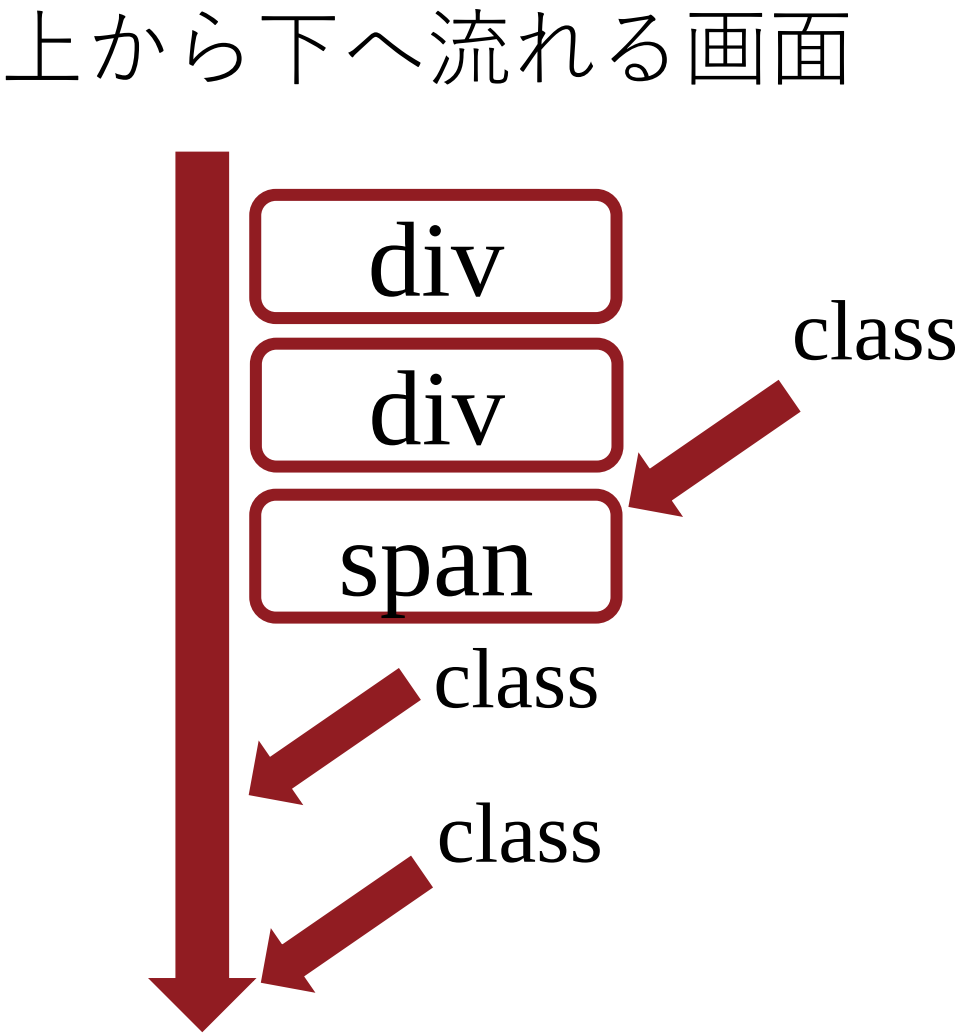
- div　　ブロック要素（段落、箱、かたまり）としてグループ化
- span　　インライン要素（文章中の一部）としてグループ化

- 違いは、前後に改行が入るかどうか

タグ関数

- 装飾ルール（style）の適用が目的であれば、
- Divやspanが範囲でくくるのとは別に、ピンポイントで選んでグループ化することもできる

- それが classという属性の利用



タグ関数

- 例えば
  p(“本文”, class=“font-red”)
  と指定して、font-redというclassのstyleを別で指定する

- Rで言えば自作関数を定義して呼び出すことに似ている

- こうすることで、
  複雑なお手製のstyleを簡単に使い回せる

タグ関数

- アプリを起動して、実装例を見てみましょう