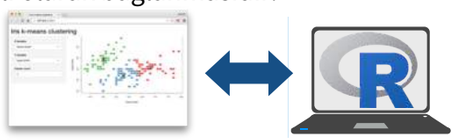
# Shiny : : HATIRLATICI NOT

## Temel Öğeler

**Shiny** uygulaması bir web sayfasının (**UI**) bilgisayarda çalışan bir R oturumuna (**Server**) canlı olarak bağlanmasıdır.



Kullanıcılar UI'yı manipüle edebilir, bu server'ın UI görümününün (R kodunun çalışmasıyla) güncellemesine yol açacaktır.

### UYGULAMA ŞABLONU

Bu şablonla yeni bir uygulama yazılmaya başlanabilir. Kodu R konsolunda çalıştırarak uygulama önizlemesi elde edebilirsiniz.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** - uygulama için HTML kullanıcı arayüzünü bir araya getiren içiçe geçmiş R fonksiyonları
- **server** - UI'da gösterilen R objelerinin nasıl inşa edildiğine yönelik açıklamalar bulunan R fonksiyonu
- **shinyApp** - ui ve server'i tek bir app içinde birleştirir. Eğer kaynağı olan bir kodtan ya da fonk. içinden çağırılıyorsa, **runApp()** kullanın.

### UYGULAMANIZI PAYLAŞIN


Uygulamayı paylaşmanın en kolay yolu, RStudio'nun bir cloud servisi olan shinyapps.io adresine koymaktır.

1. Ücretsiz ya da profesyonel bir hesap açın, **http://shinyapps.io**
2. RStudio ide'sindeki **Publish** düğmesine tıklayın ya da aşağıdaki satırı çalıştırın: **rsconnect::deployApp("<yürütülen dizin>")**

**Kendi Shiny Server'ınızı kurun**

at www.rstudio.com/products/shiny-server/

## Uygulama Kurmak

Aşağıdaki şablonu fluidPage() ve server fonk.nun gövde kısmına argümanlar ekleyerek tamamlayın.

*Input() fonksiyonlarıyla girdileri UI'ye ekleyin

*Output() fonksiyonlarıyla çıktıları ekleyin

Server'a çıktıları R'nin server fonksiyonuyla nasıl işleyeceğini söyleyin. Bunu yapmak için:

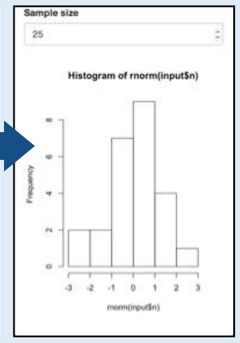
1. Çıktıları **output$<id>**
2. Girdileri **input$<id>**
3. Kodu, çıktıya kaydetmeden önce, **render*()** fonks. ile tamamlayın

```
library(shiny)

ui <- fluidPage(
  numericInput(  inputId = "n"  ,
    "Sample size", value =  25),
  plotOutput(  outputId = "hist"  )
)

server <- function(input, output) {
   output$hist  <-  renderPlot  ({
   hist(rnorm(  input$n  ))
  })
}

shinyApp(ui = ui, server = server)
```




Şablonu **app.R** isminde kaydedin. Alternatif olarak, şablonu **ui.R** ve **server.R** olarak iki dosyaya ayırabilirsiniz.

```
library(shiny)

ui <-  fluidPage(
  numericInput(  inputId = "n"  ,
    "Sample size", value =  25),
  plotOutput(  outputId = "hist"  )
)

server <-  function(input, output) {
   output$hist  <-  renderPlot  ({
   hist(rnorm(  input$n  ))
  })
}

shinyApp(ui = ui, server = server)
```

```
# ui.R

fluidPage(
  numericInput(  inputId = "n"  ,
    "Sample size", value =  25),
  plotOutput(  outputId = "hist"  )
)
```

**ui.R** ui'ye kaydedebileceğiniz her şeyi kapsar.

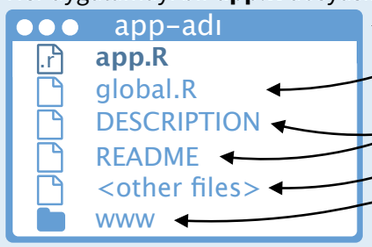
```
# server.R

function(input, output) {
   output$hist  <-  renderPlot  ({
   hist(rnorm(  input$n  ))
  })
}
```

**server.R** server'a kaydedebileceğiniz fonksiyon ile biter.

Çağırmaya gerek yoktur **shinyApp()**.

Her uygulamayı bir **app.R** dosyasını tutan (ya da bir **server.R** ve **ui.R** dosyası) dizin olarak kaydedin.




Uygulamanın (app) adı dizin adıdır

(opsiyonel) hem ui.R hem de server.R'ye uygun olan objeleri tanımlar

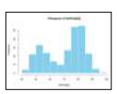
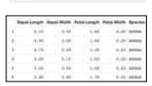
(opsiyonel) showcase modunda kullanılır

(opsiyonel) data, scripts, vb.

(opsiyonel) web tarayıcılarıyla (resimler, CSS, .js, vb.) dosyaların dizinini paylaşmak. Adı "**www**" olmalıdır

Uygulamayı başlatmak runApp(<yürütülen dizin>)

## Çıktılar - render*() ve *Output() fonks.ları UI'ya çıktı eklemek için birlikte çalışırlar

| | birlikte çalışır | |
|---|---|---|
|  | DT::**renderDataTable**(expr, options, callback, escape, env, quoted) | **dataTableOutput**(outputId, icon, …) |
| | **renderImage**(expr, env, quoted, deleteFile) | **imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId) |
|  | **renderPlot**(expr, width, height, res, …, env, quoted, func) | **plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId) |
|  | **renderPrint**(expr, env, quoted, func, width) | **verbatimTextOutput**(outputId) |
|  | **renderTable**(expr,…, env, quoted, func) | **tableOutput**(outputId) |
| foo | **renderText**(expr, env, quoted, func) | **textOutput**(outputId, container, inline) |
|  | **renderUI**(expr, env, quoted, func) | & **uiOutput**(outputId, inline, container, …) **htmlOutput**(outputId, inline, container, …) |

## Girdiler

kullanıcıdan değerler toplar

Girdinin şu anki değerine **input$<inputId>** ile erişin. Input değerleri **reaktif**tir.

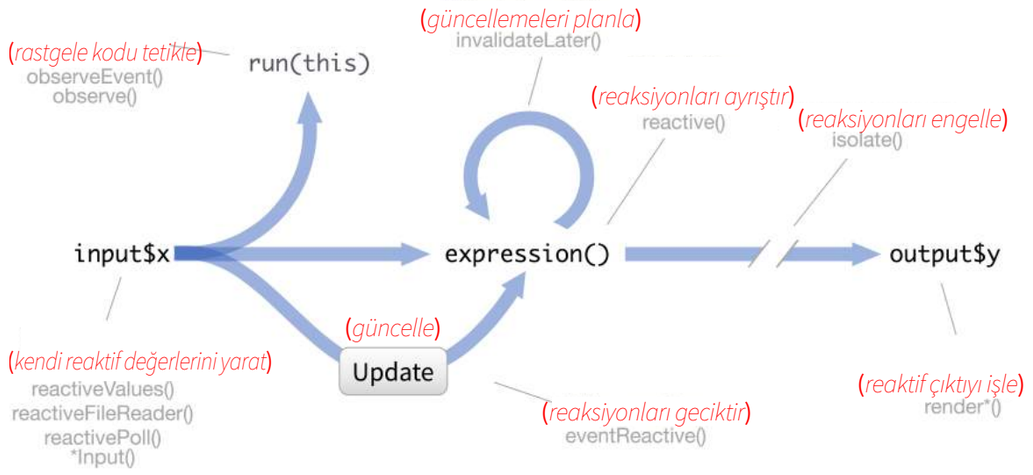
| | |
|---|---|
| Action | **actionButton**(inputId, label, icon, …) |
| Link | **actionLink**(inputId, label, icon, …) |
| Choice 1<br>Choice 2<br>Choice 3<br>Check me | **checkboxGroupInput**(inputId, label, choices, selected, inline)<br>**checkboxInput**(inputId, label, value) |
|  | **dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)<br>**dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator) |
| Choose File | **fileInput**(inputId, label, multiple, accept) |
| 1 | **numericInput**(inputId, label, value, min, max, step) |
| ••••••• | **passwordInput**(inputId, label, value) |
| Choice A<br>Choice B<br>Choice C | **radioButtons**(inputId, label, choices, selected, inline) |
| Choice 1<br>Choice 1<br>Choice 2 | **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (ayrıca selectizeInput()) |
| 0 5 10<br>0 2 4 6 8 10 | **sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post) |
| Apply Changes | **submitButton**(text, icon)<br>(Uygulamada reaksiyon oluşmasını önler) |
| Enter text | **textInput**(inputId, label, value) |



Çeviri: Metin Yazici • *yyazicimetin@gmail.com*

# Reaktiflik

Reaktif değerler reaktif fonksiyonlarla birlikte çalışır. Hatadan kaçınmak için reaktif bir değeri aşağıdaki fonksiyon argümanlarının içinde bulunanların birinden çağırın. Aktif bir reaktif içerik haricinde operasyon'a izin verilmemektedir.




## KENDİ REAKTİF DEĞERLERİNİ YARAT

```
# example snippets


ui <- fluidPage(
 textInput("a","","A")
)


server <-
function(input,output){
  rv  <-     reactiveValues        ( )
 rv$number <- 5
}
```

### *Input() functions
(ilk sayfaya bakın)

### reactiveValues(...)

Her girdi fonksiyonu **input$<inputId>** olarak saklanan bir reaktif değer yaratır.

**reactiveValues()** değerlerini belirleyebileceğin reaktif değerlerin bir listesini çıkarır.

## REAKTİF ÇIKTIYI İŞLE

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A")                ,
 textOutput("b")
)

server <-
function(input,output){
  output$b    <-
    renderText({
  input$a
    })
}

shinyApp(ui, server)
```

### render*() functions
(ilk sayfaya bakın)

Göstermek için bir obje oluşturur. Kodun içindeki herhangi bir reaktif değer değiştiğinde, kodu yeniden gövde içinden objeyi yeniden inşa etmek için tekrardan çağırır.

Sonuçları kaydetmek için: **output$<outputId>**

## REAKSİYONLARI ENGELLE

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A")                ,
 textOutput("b")
)

server <-
function(input,output){
  output$b    <-
  renderText({
      isolate({input$a})
  })
}

shinyApp(ui, server)
```

### isolate(expr)

Kod bloku çalıştırır. Sonuçların **reaktif olmayan** bir kopyasını getirir.

## RASTGELE KODU TETİKLE

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A")                ,
 actionButton("go","Go")
)

server <-
function(input,output){
  observeEvent(input$go,{
  print(input$a)
    })
}


shinyApp(ui, server)
```

### observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, labe, suspended, priority, domain, autoDestroy, ignoreNULL)

İlk argümandaki reaktif değerler değiştiğinde, ikinci argümandaki kodları çalıştırır. **observe()** ile alternatiflere bakınız.

## REAKSİYONLARI AYRIŞTIR

```
ui <- fluidPage(
 textInput("a","","A")                ,
 textInput("z","","Z"),
 textOutput("b"))

server <-
function(input,output){
  re <-     reactive({

paste(input$a,input$z)})
  output$b    <-    renderText({
      re()
  })
}
shinyApp(ui, server)
```

### reactive(x, env, quoted, label, domain)

Bir **reaktif ekspresyon** yaratır

- bilgisayar hesaplamasını azaltmak için değerleri depolar
- başka kodlardan çağırılabilir
- iptal edildiğinde bağımlı olduğu kaynaklar için uyarı verir

Ekspresyonları fonksiyon uzantısı, mesela re(), ile çağırır

## REAKSİYONLARI GECİKTİR

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A")                ,
 actionButton("go","Go"),
 textOutput("b")
)

server <-
function(input,output){
  re <-     eventReactive(
 input$go,{input$a})
  output$b    <-    renderText({
      re()
  })
}

shinyApp(ui, server)
```

### eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL)

Reaktif değerler ilk argümanda değişip ikinci argüman geçersiz kılındığında reaktif ekspresyon kodu yaratır.

# UI - Bir uygulamanın UI'si HTML belgesidir

Bu HTML'yi düzenlemek için R-Shiny fonk. kullanın.

```
fluidPage(
 textInput("a","")
)
```

HTML'ye dönüştürür

```
##   <div class="container-fluid">
## <div class="          form-group shiny-input-container                ">
##     <label for="a"></label>
##   <input id="a" type="text"
##                class="form-control" value=""/>
##    </div>
##  </div>
```

Statik HTML elemanlarını tags ile oluşturun. Tags HTML tagleri ile paralel giden ortak fonksiyonların listesidir, misal tags$a(). İsimsiz argümanlar tagın içine kaydırılacaktır; isimlendirilmiş argümanlar ise tag özelliklerine haline gelecektir.

| | | | | |
|---|---|---|---|---|
| tags$a | tags$data | tags$h6 | tags$nav | tags$span |
| tags$abbr | tags$datalist | tags$head | tags$noscript | tags$strong |
| tags$address | tags$dd | tags$header | tags$object | tags$style |
| tags$area | tags$del | tags$hgroup | tags$ol | tags$sub |
| tags$article | tags$details | tags$hr | tags$optgroup | tags$summary |
| tags$aside | tags$dfn | tags$HTML | tags$option | tags$sup |
| tags$audio | tags$div | tags$i | tags$output | tags$table |
| tags$b | tags$dl | tags$iframe | tags$p | tags$tbody |
| tags$base | tags$dt | tags$img | tags$param | tags$td |
| tags$bdi | tags$em | tags$input | tags$pre | tags$textarea |
| tags$bdo | tags$embed | tags$ins | tags$progress | tags$tfoot |
| tags$blockquote | tags$eventsource | tags$kbd | tags$q | tags$th |
| tags$body | tags$fieldset | tags$keygen | tags$ruby | tags$thead |
| tags$br | tags$figcaption | tags$label | tags$rp | tags$time |
| tags$button | tags$figure | tags$legend | tags$rt | tags$title |
| tags$canvas | tags$footer | tags$li | tags$s | tags$tr |
| tags$caption | tags$form | tags$link | tags$samp | tags$track |
| tags$cite | tags$h1 | tags$mark | tags$script | tags$u |
| tags$code | tags$h2 | tags$map | tags$section | tags$ul |
| tags$col | tags$h3 | tags$menu | tags$select | tags$var |
| tags$colgroup | tags$h4 | tags$meta | tags$small | tags$video |
| tags$command | tags$h5 | tags$meter | tags$source | tags$wbr |

En yaygın taglar wrapper fonksiyonlarına sahip olanlardır. Bunların başına **tags$** eklemek zorunda değilsiniz.

```
ui <- fluidPage(
 h1("Header 1")          ,
   hr()   ,
   br()   ,
   p( strong("bold")      ),
   p( em("italic")        ),
   p( code("code")        ),
 a(href="", "link"),
   HTML("<p>Raw html</p>")
)
```


CSS dosyası eklemek için, **includeCSS()** ya da
1. Dosyayı **www** altdizinine yerleştirin
2. Alttaki kod ile bağlayın,

```
tags$head(tags$link(rel = "stylesheet",
 type = "text/css", href = "                <file name>      " ))
```

JavaScript eklemek için, **includeScript()** ya da
1. Dosyayı **www** altdizinine yerleştirin
2. Alttaki kod ile bağlayın,

```
tags$head(tags$script(src = "<file name>                    "))
```

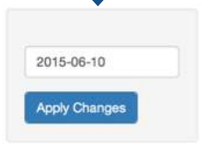
**IMAGES**

Resim eklemek için
1. Dosyayı **www** altdizinine yerleştirin
2. Bu kod ile bağlayın, img(src="<file name>")

# Tasarım-Düzen

Çeşitli elemanları kendi panel fonksiyon ile özellikleri olan tek bir elemanda birleştirin. Örneğin,

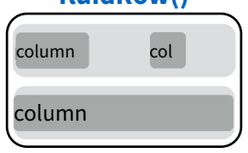
```
wellPanel(      dateInput("a", ""),
      submitButton()
)
```




| | |
|---|---|
| absolutePanel() | navlistPanel() |
| conditionalPanel() | sidebarPanel() |
| fixedPanel() | tabPanel() |
| headerPanel() | tabsetPanel() |
| inputPanel() | titlePanel() |
| mainPanel() | wellPanel() |

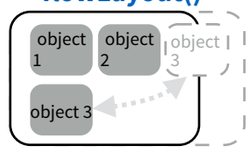
Panelleri ve elemanları bir layout fonksiyonu ile tasarım içine yerleştirin. Elemanları layout fonk.larının argümanları olarak ekleyin.

### fluidRow()



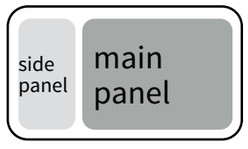

```
ui <- fluidPage(
 fluidRow(column(width = 4),
   column(width = 2,  offset = 3)),
 fluidRow(column(width = 12))
)
```

### flowLayout()



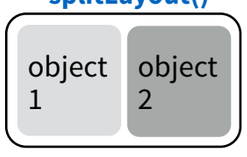

```
ui <- fluidPage(
 flowLayout( # object 1,
             # object 2,
             # object 3
 )
)
```

### sidebarLayout()



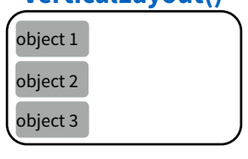

```
ui <- fluidPage(
 sidebarLayout(
   sidebarPanel(),
   mainPanel()
 )
)
```

### splitLayout()



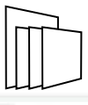

```
ui <- fluidPage(
 splitLayout( # object 1,
              # object 2
 )
)
```

### verticalLayout()




```
ui <- fluidPage(
 verticalLayout( # object 1,
                 # object 2,
                 # object 3
 )
)
```

tabPanelleri birbirinin üzerine katmanlayın ve birbirleri arasında geçiş yapın.



```
ui <- fluidPage( tabsetPanel(
   tabPanel("tab 1", "contents"),
   tabPanel("tab 2", "contents"),
   tabPanel("tab 3", "contents")))
```


```
ui <- fluidPage( navlistPanel(
   tabPanel("tab 1", "contents"),
   tabPanel("tab 2", "contents"),
   tabPanel("tab 3", "contents")))
```


```
ui <- navbarPage(title = "Page",
   tabPanel("tab 1", "contents"),
   tabPanel("tab 2", "contents"),
   tabPanel("tab 3", "contents"))
```




Çeviri: Metin Yazici • *yyazicimetin@gmail.com*