# R language and data analysis: function

Qiang Shen

Jan. 9, 2018

# R programmer

- R User: SPSS

# R programmer

- R User: SPSS
- R Programmer/developer: python,C

# R programmer

- R User: SPSS
- R Programmer/developer: python,C
- Pareto principle (80/20 rule)

# R programmer

- R User: SPSS
- R Programmer/developer: python,C
- Pareto principle (80/20 rule)
- problem solving

# functional programming

- Functional Programming (FP)
- object oriented programming (OOP)

# functional programming

Functions are **first class** objects

- treated like any other R object
- pass as arguments to other functions (apply)
- can be nested inside another function

# advantage of function

- reusability (copy and paste)
- less mistake prone.
- generalization

# function

What is function?

- name
- arguments
- function content
- return
- enviorment

# function

```
func = function (optional_arguments){
    interesting statement(s)
}
```

# most simplest function

```
f<-function(){
  }
class(f)
```

```
[1] "function"
```

```
f()
```

```
NULL
```

# How to define function

$$x^2 + 5x + 2$$

```
func <- function (x) {
    x^2 + 5*x + 2
}
func(3)
```

# How to define function

```
f<-function(num){
  hello<-'Hello World!\n'
  for (i in 1:num){
    value<-cat(hello)
  }
}
data<-f(3)
```

```
Hello World!
Hello World!
Hello World!
```

# function

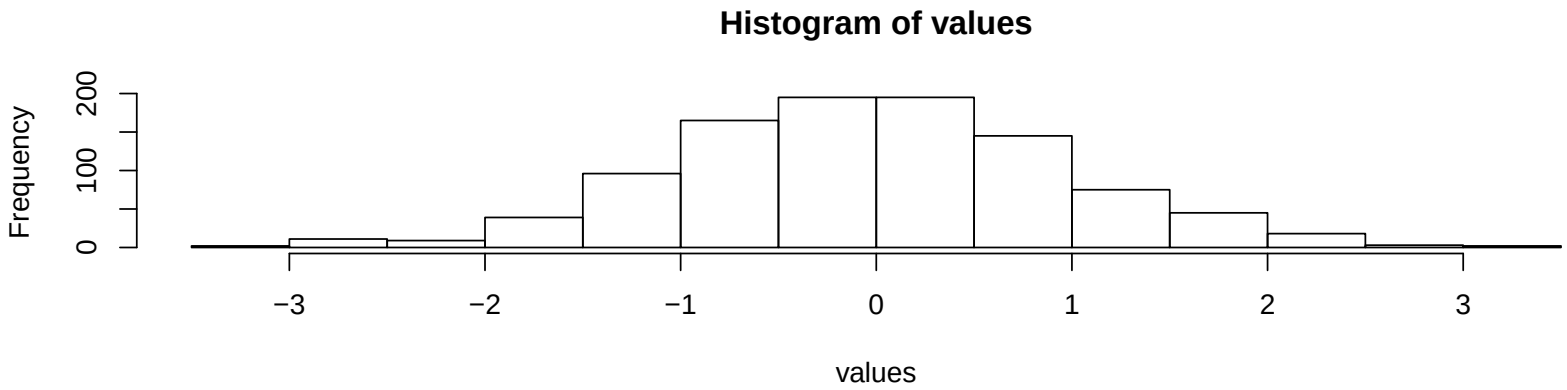
- formal argument: num
- default value
- function is to print ...
- returns the print content

```
f<-function(num=3){
  hello<-'Hello World!\n'
  for (i in 1:num)){
    value<-cat(hello)
  }
}
f(num=2)
```

# invisible return

```
histnormal = function(n) {
  set.seed(1234)
  values = rnorm(n);hist(values)
   max(values)
   # return(values)
    invisible(max(values))
            }# max(values)

a<-histnormal(1000)
```



# argument matching

```
func = function(num, benchmark=0, multiplier=3) {
  if (num < benchmark) {
    return(num / multiplier)
  } else {
    return(num * multiplier)
  }
}
func(7)
```

```
[1] 21
```

# argument matching

position matching by default

```
func(7, 3)
```

```
[1] 21
```

```
func(7, 3, 4)
```

```
[1] 28
```

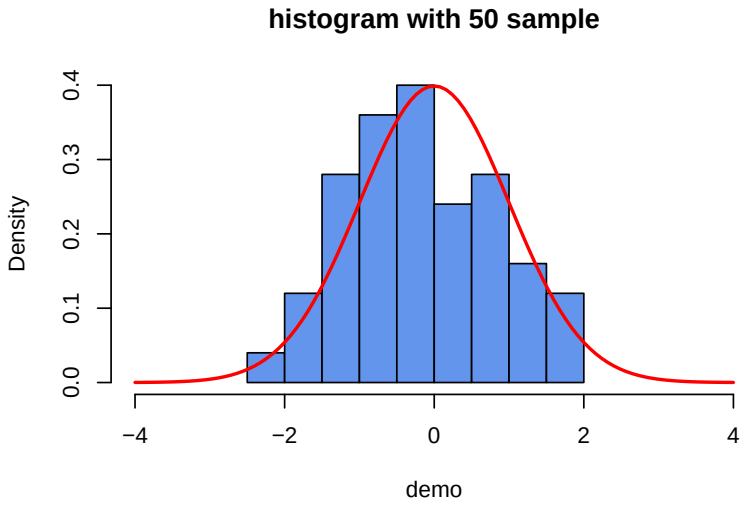
# argument matching

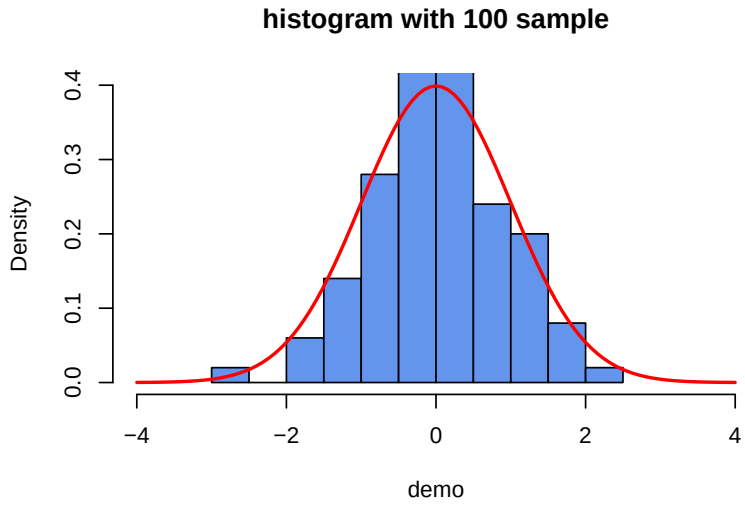
- 1.exact matching
- 2.partial matching
- 3.position matching

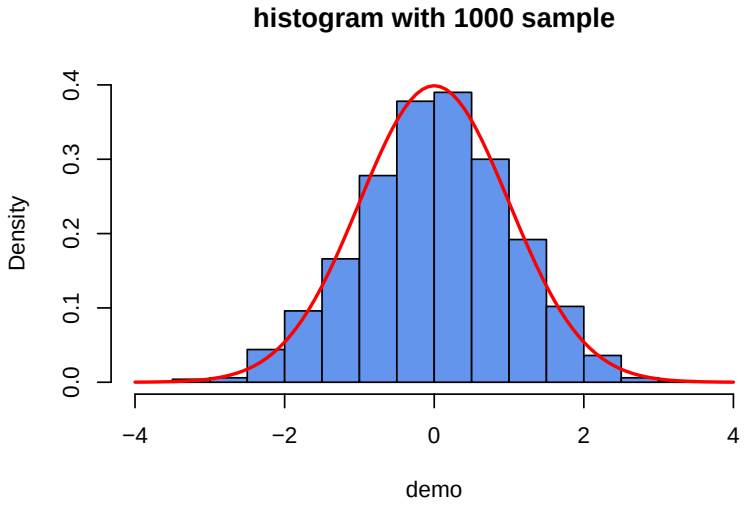
```
func(7, multiplier=3,benchmark=4)
```

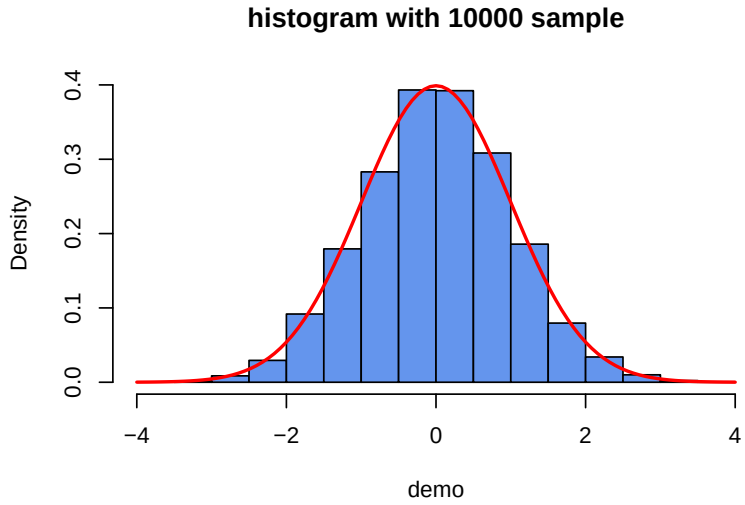
# functional programming

work together with `apply` family function.



# functional programming

work together with apply family function.

```
par(mfrow=c(2,2))
func<-function(x){
demo<-rnorm(x,mean=0,sd=1)
hist(demo,freq=FALSE,col="cornflowerblue",
     main=paste("histogram with",x,'sample'),xlim=c(-4,4),y
curve(dnorm(x), add=TRUE, col="red", lwd=2)
}
# func(50)
sapply(c(50,100,1000,1000000),func)
```

# Anonymous function

```
sapply(iris[,1:4],
 function(x) {sd(x,na.rm=T)/mean(x,na.rm=T)})
```

```
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
   0.1417113    0.1425642    0.4697441    0.6355511
```

# lazy function

```
f<- function(x,y){
                x^2
}
f(3)
```

```
[1] 9
```

```
f(3,4)
```

```
[1] 9
```

# lazy function

```
f<- function(x,y){
                print(x^2)
                print(y^3)
}
f(3)
f(3,4)
```

# The ... argument

```
with(mtcars,plot(mpg,hp))
myplot<-function(x,y, col='red',pch=1,...){
  plot(x,y,col=col,pch=pch,...)
}
with(mtcars,myplot(mpg,hp))
```

# argument after ...

exact matching vs.partial matching

```
args(paste)
args(cat)
paste('3','4 = 7',sep=" + ")
paste('3','4 = 7',se=" + ")
```

# local and global variable

```
myFun <- function (x) {
     u=2
    cat ("u=", u, "\n")    # this variable is local !
    u=u+1          # local
    cat ("u=", u, "\n")
}

# u = 2
myFun(5)
u
cat ("u=", u, "\n")

myFun <- function (x) {
     cat ("u=", u, "\n")        # this variable is local !
     u <<- u+1          # this WILL affect the value of variable
     cat ("u=", u, "\n")
}
```

# look at source code

- generic function: print,summary,plot

```
lm
methods("summary")
summary.data.frame
```

# execute a function from file

source ("file", ... )

```
# load foo.r source file
source ("SS.r")
SS
```

# function structure

- name
- arguments
- function content
- return
- enviorment