#Slide # 7, #Slide # 8

help.start()

?mean

help(mean) # same

?`if`

?iris

example(paste)

#The demo function gives longer demonstrations of how to use a function.

demo(plotmath)

demo(graphics)

#Finding a function that you don't know the name of

??regression

help.search("regression")

#Searches several sites directly from R.

RSiteSearch("matrix")

#Slide # 12

#factor example

x <- factor(c("single", "married", "married", "single"));

#Slide # 13

#Another example of Factor:

data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)

fdata = factor(data)

fdata

# nominal

rdata = factor(data,labels=c("I","II","III"))

# ordinal

rdata = factor(data, ordered=T, labels=c("I","II","III"))

rdata

#To convert the default factor fdata to roman numerals, we use the assignment form of the levels function:

levels(fdata) = c('I','II','III')

fdata

#String – Any value written inside a single quote or double quotes is referred to as String.

#For example,

x <- "This is a valid proper ‘ string"

#Slide # 15 Variables in R

# Assignment using equal operator.

var.1 = c(0,1,2,3)

# Assignment using leftward operator.

var.2 <- c("learn","R")

# Assignment using rightward operator.

c(TRUE,1) -> var.3

print(var.1)

cat ("var.1 is ", var.1 ,"\n")

cat ("var.2 is ", var.2 ,"\n")

cat ("var.3 is ", var.3 ,"\n")

#Slide # 17 Scalars - Number

x <- 1

y <- 2.5

class(x)

class(y)

class(x+y)

#Slide # 18 Scalars - Logical value

m <- x > y # Is x larger than y?

n <- x < y # Is x smaller than y?

m

n

class(m)

class(NA) # NA is another logical value: 'Not Available'/Missing Values

m & n # AND

m | n # OR

!m # Negation

#Slide # 19 Scalars - Character(string)

a <- "1"; b <- "2.5" # Are they different from x and y we used earlier?

a;b

a+b # a+b=3.5? #Error in a + b : non-numeric argument to binary operator

class(a)

class(as.numeric(a)) # but you can coerce this character into a number

x <- 6

class(as.character(x)) # vice resa

#Slide # 20 Vectors

#A vector is a sequence of data elements of the same basic type.

o <- c(1,2,5.3,6,-2,4) # Numeric vector

p <- c("one","two","three","four","five","six") # Character vector

q <- c(TRUE,TRUE,FALSE,TRUE,FALSE,TRUE) # Logical vector

o;p;q

#Slide # 21 Vectors

o[q] # Logical vector can be used to extract vector components

names(o) <- p # Give each component a name

o

o["three"] # Extract your components by "calling" their names

#Slide # 22 Matrices

#A matrix is a collection of data elements arranged in a two-dimensional rectangular layout.

#Same as vector, the components in a matrix must be of the same basic type.

#The following is an example of a matrix with 4 rows and 3 columns.

t <- matrix(1:12,nrow=4,ncol=3,byrow = FALSE)

t

#Slide # 23 Matrices

#Similar to vectors, matrices also use [] to reference elements.

t[2,3] # component at 2nd row and 3rd column

t[,3] # 3rd column of matrix

t[4,] # 4th row of matrix

t[2:4,1:3] # rows 2,3,4 of columns 1,2,3

#Slide # 24 Data Frames

#A data frame is more general than a matrix, in that different columns can have different basic data types.

d <- c(1,2,3,4)

e <- c("red", "white", "red", NA)

f <- c(TRUE,TRUE,TRUE,FALSE)

mydata <- data.frame(d,e,f)

names(mydata) <- c("ID","Color","Passed") # variable names

mydata

#Slide # 25 Data Frames

#Extracting components from data frames is somehow similar to what we did for matrices, but after assigning names to each column (variable), it becomes more flexible.

mydata$ID # try mydata["ID"] or mydata[1]

mydata$ID[3] # try mydata[3,"ID"] or mydata[3,1]

mydata[1:2,] # first two records

#Slide # 26 List

#A list is a generic vector containing other objects.

#There is no restriction on data types or length of the components.

# a list with a vector, a matrix, a data frame defined earlier and a scalar

p=c("one", "two", "three", "four", "five", "six")

l <- list(vec=p, mat=t, fra=mydata, count=3)

l

#Slide # 27 List

# extract components from list

l$vec

l$mat[2,3]

l$fra$Color