```
"What's up, guys!"
5 + 5
#we can directly type in something to the console
#(just like MATLAB)

#variable assignment
name <- "Yusen"
nchar(name)
#The length of the string
grepl("H", name)
grepl("Y", name)
#contains method in R: grepl()

age <- 19
name #just type in, we get the output
print(age) #alternative way to output

paste("My age is", age)
#very useful to concatenate everything
#concatenate using paste() function, separated by commas

#class() : get the data type
class(name)
#data type: character
class(age)
#data type: numeric
x <- -9i + 1
class(x)
#data type: complex

my_year <- 2L #cap le
class(my_year)
#data type: integer

isVisited = TRUE
class(isVisited)
#data type: logical

#there are three types of number values
#numeric, integer, and complex

#type casting
x <- 1L # integer
y <- 2 # numeric

# convert from integer to numeric:
a <- as.numeric(x)

# convert from numeric to integer:
b <- as.integer(y)

min(3,2,5)
max(3,2,5)
sqrt(81)
abs(-10)
ceiling(1.4) # round up
floor(1.4) # round down

2^3
#exponent
10 %% 2
#modular
10 %/% 2
#integer division

#logical operator: same as Java

a <- 2
b <- 3
if(a < b){
```

```
  print("smaller!")
}else if(a > b){
  print("larger!")
}else{
  print("shoot! equal")
}

#loop: no more continue: we use "next" instead.

#loop:
for(x in 1:10){
  print(x)
}
#caveat: the right-end point is also inclusive

#declare functions
#declaration and implementation
sumOfDigit <- function(number){
  sum <- 0
  if(number >= 10){
    sum <- sumOfDigit(number %/% 10) + number %% 10
  }else{
    sum <- number
  }
  return (sum)
  #the parentheses are required
}

#function call
sumOfDigit(543211234512345)

text1 <- "this is a local variable"
text2 <<- "this is is a global variable"
#global declaration

#various data structure
#vector: a list of the same data type
fruits <- c("banana", "apple", "orange")
#a vector of strings
numbers <- c(1, 2, 3)
#a vector of numerics
numbers <- 1:10
#a vector of numerics in sequence
numbers1 <- 1.5:6.5
#it also works with demicals
number2 <- seq(from = 0, to = 100, by = 20)
#a formal way to use vector in sequence
#both "from" and "to" are inclusive
length(fruits)
#the length of the vector
sort(fruits)
#sort the vector
fruits[1]
#access items (start from 1! really sucks like MATLAB)
fruits[c(1,3)]
#access multiple items
fruits[c(-1)]
# Access all items except for the first item
fruits[1] <- "pear"
#change an item

#Lists: a list of different data types
thislist <- list("apple", "banana", "cherry")
#a list of strings
"apple" %in% thislist
"Apple" %in% thislist
#check if item exists: "%in%" membership operator
#return TRUE  FALSE

thislist <- append(thislist, "orange")
```

```
thislist
#append items to the end
#caveat: must REASSIGN it to the original list
thislist <- append(thislist, "grape", after = 0)
thislist
#specify the index (insert items AFTER a particular index position)
thislist <- thislist[-1]
thislist
#remove items (just pass in the negative sign to remove)
#keep in mind: always reassign it to the original list
(thislist)[2:4]
#range of indexing

for(x in thislist){
  print(x)
}
#loop through a list

list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)
#using c() command to join/concatenate two lists
list3

#Matrices
thismatrix <- matrix(c(1,6,5,3,2,5,5,5), nrow = 4, ncol = 2)
#create a matrix
#note: fill vertically (column-wise) first, then fill out another column
thismatrix
thismatrix[1,2]
#access the item: mat[#row, #col]
thismatrix[2,]
#the entire second row
thismatrix[,2]
#the entire second column (just like MATLAB)
thismatrix[c(1,2),]
#the first two rows
thismatrix <- cbind(thismatrix, c(3,6,1,8))
#add columns: cbind() command
thismatrix <- rbind(thismatrix, c(4,5,6))
#add rows: rbind() command
thismatrix

thismatrix <- thismatrix[-c(1), -c(1)]
thismatrix
#remove the first row and the first column
9 %in% thismatrix
#check if an item exists
dim(thismatrix)
#output the dimension of the matrix format: #row  #col
length(thismatrix)
#output the total number of cells in the matrix

for (r in 1 : nrow(thismatrix)) {
  for (c in 1 : ncol(thismatrix)) {
    print(thismatrix[r, c])
  }
}

# Combine matrices
Matrix1 <- matrix(c("apple", "banana", "cherry", "grape"), nrow = 2, ncol = 2)
Matrix2 <- matrix(c("orange", "mango", "pineapple", "watermelon"), nrow = 2, ncol = 2)
Matrix_Combined <- rbind(Matrix1, Matrix2)
#concatenate as rows (vertically)
Matrix_Combined
Matrix_Combined <- cbind(Matrix1, Matrix2)
#concatenate as columns (horizontally)
Matrix_Combined

#Arrays: multi-dimensions
```

```
# Access all the items from the first row from matrix one
multiarray <- array(c(1:24), dim = c(4, 3, 2))
# Parameter: dim = c(#row, #col, #dimension)
# multiarray
dim(multiarray)
#output each dimension respectively
length(multiarray)
#output the total number of cells
for(x in multiarray){
  print(x)
}
#loop through an array

#Data Frames
#different data types in a table
#each column should have the same type of data
my_dataframe <- data.frame(
  language = c("C/C++", "Java", "Python"),
  fluency = c(60,100,70),
  frequency = c(10,80,10)
  #specify each column in the dataframe
)
my_dataframe
summary(my_dataframe)

my_dataframe[1]
#access the first column
my_dataframe["language"]
#alternatively, we can specify the name of the column to access it
my_dataframe$language
#alternative: only get the content

my_dataframe <- rbind(my_dataframe, c("R", 50, 5))
#add a new row
my_dataframe

my_dataframe <- my_dataframe[-c(2),]
#remove the second row
my_dataframe
dim(my_dataframe)
#output the dimension of the dataframe
length(my_dataframe)
#output the total number of columns!!!!!
#different from previous data structures

#Factors
#Factors are used to categorize data.
coding_genre <- factor(c("C", "C++", "Java", "JavaScript", "Python", "C++", "R", "Ruby", "MATLAB"))
#create a factor
coding_genre
#print the factor
levels(coding_genre)
#only print out the levels in the factor
length(coding_genre)
#the total number of items in the factor(include duplicates)

coding_genre[6] <- "C"
#modify items in the factor
#caveat: the incoming value must be pre-defined in the factor
coding_genre[6]


#R graphics:
#plot command
#by default, R graphics represent scatter plots
x_val <- c(1,2,3,4,5,6,7,8,9,10)
y_val <- c(1,3,5,6,7,8,2,3,9,5)

plot(x_val, y_val, main="my first graph", xlab="x values", ylab="y values", col="purple")
#type: the type of represenatation
```

```
#main: title
#xlab: x-axis label
#ylab: y-axis label
#col: displayed color

x_arr <- 1:10
y_arr <- c(10,9,8,7,6,5,4,3,2,1)
plot(x_arr, y_arr, cex=2, pch=10, col="blue")
#cex: the size of data points (default value: 1)
#pch: the shape of data points (from 0 to 25)
#superimpose: use points()instead
y_arr2 <- c(2,3,5,8,7,4,6,1,9,10)
points(x_arr, y_arr2,cex=2, pch=15, col="pink")


#line graphs
#plot(1:10, type="l", col="blue", lwd=2, lty=6)
#type is specified as "l" (line)
#lwd: line width (default value: 1)
#lty: line type (from 0 to 6)
#superimpose: use lines()command
#lines(c(1,3,5,7,9,2,4,6,8,10), type="l",col="red", lwd=3, lty=3)


#pie chart
my_pie <- c(10,20,30,40)
pie(my_pie,init.angle=90)
#init,angle: the initial angle of the first pie in degrees

my_label <- c("Apples", "Bananas", "Cherries", "Dates")
#label array
my_color <- c("red", "yellow", "pink", "purple")
#color array
pie(my_pie, label=my_label, col=my_color, main="fruit glosory")

legend("bottomright", my_label, fill=my_color)
#add legend to the pie chart


#bar chart
x <- c("A", "B", "C", "D")
y <- c(2, 4, 10, 7)
barplot(y, names.arg=x, density=5)
#density: the quantity that changes bar's texture
barplot(y, names.arg=x, density=5,horiz=TRUE)
#horizontal bar chart
mtcars
#a built-in dataset
#?mtcars
#get information about the dataset
Data_Cars <- mtcars
dim(Data_Cars)
#dimension of the given dataset
names(Data_Cars)
#name of each variable
rownames(Data_Cars)
#The name of each row
Data_Cars$gear
Data_Cars$mpg
#print variable values
sort(Data_Cars$cyl)
#sort variable values
summary(Data_Cars)
#output summary for each variable in six statistics:
#Min, 1st Qu, median, mean, 3rd Qu, Max
max(Data_Cars$hp)
#find the max value
which.max(Data_Cars$hp)
#find the index position of the max value
rownames(Data_Cars)[which.max(Data_Cars$hp)]
```

```
#find the item which has the max value
mean(Data_Cars$gear)
median(Data_Cars$gear)
#find mean and median
names(sort(-table(Data_Cars$gear)))[1]
#find mode
quantile(Data_Cars$wt,0.75)
#calculate quantile
```