Problem Statement 1: Demonstrate Various Types of Command for Help in R.

Objective: To demonstrate various types of command for Help in R.

Theory: R includes extensive facilities for accessing documentation and searching for help. There are also specialised search engines for accessing information about R on the internet, and general internet search engines can also prove useful.

Command + Output (code starts with ">"):

> help("mean")

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm	a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.
...	further arguments passed to or from other methods.

> help.search("mean")

Help pages:

base::colSums	Form Row and Column Sums and Means
base::Date	Date Class
base::DateTimeClasses	Date-Time Classes
base::difftime	Time Intervals / Differences
base::mean	Arithmetic Mean
boot::sunspot	Annual Mean Sunspot Numbers
cluster::meanabsdev	Internal cluster functions
lattice::tmd	Tukey Mean-Difference Plot
Matrix::Matrix-class	Virtual Class "Matrix" Class of Matrices

> apropos("mea")

```
 [1] ".colMeans"          ".rowMeans"
 [3] "colMeans"           "influence.measures"
 [5] "kmeans"             "mean"
 [7] "mean.Date"          "mean.default"
 [9] "mean.difftime"      "mean.POSIXct"
[11] "mean.POSIXlt"       "rowMeans"
[13] "weighted.mean"
```

> RSiteSearch("mean")

mean Search

Sort by: relevance Results per Page: 20

Search in: **R** Manuals Base Packages Help Pages | **CRAN** Task Views

CRAN Packages Description Help Pages News Readme Vignettes

Matching any words Matching all words

1-20 of about 76,000 matches

Term frequencies: **mean**: 76,559

Search took 0.006771 seconds

Modified: **2022-11-26** Size: **1.8K**

R: QTS Geometric Mean

mean.qts_sample {squat} R Documentation QTS Geometric **Mean** Description This function computes the pointwise geometric **mean** of a QTS sample. Usage ## S3 method for class 'qts_sample' **mean**(x...

/CRAN/refmans/squat/html/mean.qts_sample.html

cran-help matching: *mean*, *mean* and *mean*

Modified: **2022-11-26** Size: **1.6K**

R: Mean

mean.outlier_model {molic} R Documentation **Mean** Description Estimated **mean** of deviance statistic T(Y) Usage ## S3 method for class 'outlier_model' **mean**(x, ...) Arguments x A outlier_model object ...

/CRAN/refmans/molic/html/mean.outlier_model.html

cran-help matching: *mean*, *mean* and *mean*

> help.start()

Statistical Data Analysis

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
NEWS
Authors
Frequently Asked Questions
User Manuals
Resources
Thanks
Technical papers

Material specific to the Windows port

CHANGES up to R 2.15.0
Windows FAQ

Problem Statement 2: Demonstrate Various Command for Packages in R.

Objective: To demonstrate various commands for packages in R.

Theory: Packages in R programming language are a set of R functions, compiled code, and sample data. These are stored under a directory called “library” within the R environment. By default, R installs a group of packages during installation. Once we start the R console, only the default packages are available by default. Other packages that are already installed need to be loaded explicitly to be utilised by the R program that’s getting to use them.

Command + Output:

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"   "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

```
> installed.packages()
                Package
base            "base"
boot            "boot"
class           "class"
cluster         "cluster"
codetools       "codetools"
compiler        "compiler"
datasets        "datasets"
foreign         "foreign"
graphics        "graphics"
grDevices       "grDevices"
……
```

```
> library()
Packages in library 'S:/Softwares/Scoop/apps/r/4.2.1/library':

base                The R Base Package
boot                Bootstrap Functions (Originally by
                    Angelo Canty for S)
class               Functions for Classification
cluster             "Finding Groups in Data": Cluster
                    Analysis Extended Rousseeuw et al.
codetools           Code Analysis Tools for R
compiler            The R Compiler Package
datasets            The R Datasets Package
foreign             Read Data Stored by 'Minitab', 'S',
                    'SAS', 'SPSS', 'Stata', 'Systat',
                    'Weka', 'dBase', ...
graphics            The R Graphics Package
grDevices           The R Graphics Devices and Support
                    for Colours and Fonts
```

> vignette()

```
Vignettes in package 'grid':

moveline           Demonstrating move-to and line-to
                   (source, pdf)
displaylist        Display Lists in grid (source, pdf)
interactive        Editing grid Graphics (source, pdf)
frame              Frames and packing grobs (source,
                   pdf)
grid               Introduction to grid (source, pdf)
locndimn           Locations versus Dimensions (source,
                   pdf)
sharing            Modifying multiple grobs
                   simultaneously (source, pdf)
nonfinite          Non-finite values (source, pdf)
saveload           Persistent representations (source,
                   pdf)
rotated            Rotated Viewports (source, pdf)
grobs              Working with grid grobs (source, pdf)
viewports          Working with viewports (source, pdf)
plotexample        Writing grid Code (source, pdf)
```

> demo()

```
Demos in package 'base':

error.catching     More examples on catching and
                   handling errors
is.things          Explore some properties of R objects
                   and is.FOO() functions. Not for
                   newbies!
recursion          Using recursion for adaptive
                   integration
scoping            An illustration of lexical scoping.

Demos in package 'graphics':

Hershey            Tables of the characters in the
                   Hershey vector fonts
Japanese           Tables of the Japanese characters in
                   the Hershey vector fonts
graphics           A show of some of R's graphics
                   capabilities
image              The image-like graphics builtins of R
```

Problem Statement 3: How to Use R as a calculator and store the results in R.

Objective: To Use R as a calculator and store the results in R.

Theory: R can be used as a powerful calculator by entering equations directly at the prompt in the command console. Simply type your arithmetic expression and press ENTER. R will evaluate the expressions and respond with the result.

Command + Output:

```
> 5+4
[1] 9

> 9*4
[1] 36

> 91/2
[1] 45.5

> 52%%3
[1] 1

> 71%/%4
[1] 17

> sqrt(7)
[1] 2.645751

> abs(-199)
[1] 199

> cos(12)
[1] 0.843854

> exp(5)
[1] 148.4132

# Storing result in a variable -

> a <- 9*2+4/2
> a
[1] 20

> b <- cos(11)
> b
[1] 0.004425698

> c <- a+b
> c
[1] 20.00443
```

Problem Statement 4: Demonstrate the usage of variables, constants and built-in constants in R.

Objective: To demonstrate the usage of variables, constants and built-in constants in R.

Theory:
In computer programming, a variable is a named memory location where data is stored.
Constants are those entities whose values aren't meant to be changed anywhere throughout the code.
In R, we can declare constants using the <- symbol.
R programming provides some predefined constants that can be directly used in our program, also known as built-in constants.

Command + output:

```
# Variables and constants

>  a <- 99 %/% 12
> a
[1] 8

> b <- "a"
> b
[1] "a"

> c <- "THis is just a test!!!"
> c
[1] "THis is just a test!!!"

> d <- 0/0
> d
[1] NaN

# Built-in constants

> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"

> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

> pi
[1] 3.141593
```

Problem Statement 5: Demonstrate the usage of all the operators in R

Objective: To demonstrate the usage of all the operators in R.

Theory:
R supports majorly four kinds of binary operators between a set of operands.
They are -
Arithmetic Operators
Logical Operators
Relational Operators
Assignment Operators

Command + Output:

```
# Arithmetic Operations

> 5+7
[1] 12

> 6^3
[1] 216

> 6%%4
[1] 2

> 6%/%4
[1] 1

# Logical Operations

> list1 <- c(TRUE, 0.1)
> list2 <- c(0,4+3i)
> print(list1 & list2)
[1] FALSE  TRUE

> list1 <- c(0,FALSE)
print(!list1)
[1] True True

> TRUE && FALSE
[1] FALSE

> TRUE || FALSE
[1] TRUE

# Relational Operations

> 5 < 10
[1] FALSE

> 10 != 7
```

```
[1] TRUE

# Assignment Operation

> vec1 <- c(2:5)
> vec1
[1] 2 3 4 5

> c(2:5) ->> vec2
> vec2
[1] 2 3 4 5

> vec3 <<- c(2:5)
> vec3
[1] 2 3 4 5

> vec4 = c(2:5)
> vec4
[1] 2 3 4 5
```

Problem Statement 6: How to enter numerical and text items as data in R.

Objective: To enter numerical and text items as data in R.

Theory: We can use combine command for creating data (both numerical and character type).

Command + Output:

```
> a <- c(2, 3, 4, 5, 6, 7)
> a
[1] 2 3 4 5 6 7
> b <- c(a, 1, 99, 1930)
> b
[1]    2    3    4    5    6    7    1   99 1930

> days <- c('Mon', "Tue", 'Wed', "Thurs")
> days
[1] "Mon"   "Tue"   "Wed"   "Thurs"
> days <- c(days, "Fri", 'Sat', "Sun")
> days
[1] "Mon"   "Tue"   "Wed"   "Thurs" "Fri"   "Sat"   "Sun"

> mixed <- c(b, days)
> mixed
 [1] "2"    "3"    "4"    "5"    "6"    "7"    "1"    "99"   "1930"
[10] "Mon"  "Tue"  "Wed"  "Thurs" "Fri"  "Sat"  "Sun"
```

Problem Statement 7: Demonstrate the usage of scan command.

Objective: To demonstrate the usage of scan command.

Theory: Scan command reads data into a vector or list from the console or file.

Command + Output:

```
# Default expects a real number as input

> a <- scan()
1: 1
2: 53
3: 131
4:
Read 3 items
> a
[1]   1  53 131

# Scanning strings

> a <- scan(what="character")
1: kdfok
2: test
3: ok
4:
Read 3 items
> a
[1] "kdfok" "test"  "ok"

> a <- scan(sep=",", what="character")
1: just,a,test
4:
Read 3 items
> a
[1] "just" "a"     "test"

# Scanning from file

> a <- scan(what="character", file="CCA CRA ParametricEQ.txt")
Read 63 items
> a
 [1] "Preamp:" "-4.2"    "dB"      "Filter"  "1:"      "ON"      "PK"
 [8] "Fc"          "19"      "Hz"      "Gain"    "-4.7"    "dB"      "Q"
[15] "0.09"        "Filter"  "2:"      "ON"      "PK"      "Fc"      "568"
[22] "Hz"          "Gain"    "1.2"     "dB"      "Q"       "1.78"    "Filter"
[29] "3:"          "ON"      "PK"      "Fc"      "862"     "Hz"      "Gain"
[36] "1.6"         "dB"      "Q"       "2.87"    "Filter"  "4:"      "ON"
[43] "PK"          "Fc"      "3213"    "Hz"      "Gain"    "4.0"     "dB"
[50] "Q"           "1.88"    "Filter"  "5:"      "ON"      "PK"      "Fc"
[57] "13"          "Hz"      "Gain"    "-0.6"    "dB"      "Q"       "1.10"
```

Problem Statement 8: How to show and set the default working directory in R.

Objective: To show and set the default working directory in R.

Theory:
The working directory is just a file path on your computer that sets the default location of any files you read into R, or save out of R.
You can only have one working directory active at any given time. The active working directory is called your *current* working directory.
You can also change the working directory.

Command + Output:

```
# To show current working directory

> getwd()
[1] "C:/Users/lucky"

# To change working directory

> setwd("S:/Downloads")
> getwd()
[1] "S:/Downloads"
```

Problem Statement 9: Write a R script to take input from user for addition of two numbers.

Objective: To take input from the user for the addition of two numbers.

Theory:
readline() function used to read input from the user.
as.double() function used to transform read data into a double.
print() function to print data to console.
paste() to concatenate vectors after converting to character.

R Script:

```
a <- as.double(readline(prompt = "Enter first number to add: "))
b <- as.double(readline(prompt = "Enter second number to add: "))
print(paste("The sum of entered numbers", a, "and", b, "is", a+b))
```

Output:

```
> source("~/.active-rstudio-document")
Enter first number to add: 9
Enter second number to add: 2.5
[1] "The sum of entered numbers 9 and 2.5 is 11.5"
> source("~/.active-rstudio-document")
Enter first number to add: -12
Enter second number to add: -9
[1] "The sum of entered numbers -12 and -9 is -21"
>
```

Problem Statement 10: Write a R script to input a number check if the number is positive or negative or zero.

Objective: To input a number and check if the number is positive or negative or zero.

Theory:
readline() function used to read input from the user.
as.double() function used to transform read data into a double.
print() function to print data to console.
if() function checks whether a condition is TRUE or FALSE.

R Script:

```
num = as.double(readline(prompt="Enter a number: "))

if(num > 0) {
  print("Entered number is a positive number.")
} else if (num < 0) {
  print("Entered number is a negative number.")
} else {
  print("Entered number is 0.")
}
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number: -12
[1] "Entered number is a negative number."
> source("~/.active-rstudio-document")
Enter a number: -0.5
[1] "Entered number is 0."
> source("~/.active-rstudio-document")
Enter a number: 0.5
[1] "Entered number is a positive number."
> source("~/.active-rstudio-document")
Enter a number: 0
[1] "Entered number is 0."
> source("~/.active-rstudio-document")
Enter a number: -1.1
[1] "Entered number is a negative number."
> source("~/.active-rstudio-document")
Enter a number: -0.5
[1] "Entered number is a negative number."
>
```

Problem Statement 11: Write a R script to check if the input number is odd or even.

Objective: To check if the input number is odd or even.

Theory:
readline() function used to read input from the user.
as.integer() function used to transform read data into an integer.
print() function to print data to console.
if() function checks whether a condition is TRUE or FALSE.

R Script:

```
num = as.integer(readline(prompt="Enter a number to check if its even or odd: "))

if(num %% 2 == 0) {
  print("Entered number is an even number.")
} else {
  print("Entered number is an odd number.")
}
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number to check if its even or odd: 12
[1] "Entered number is an even number."
> source("~/.active-rstudio-document")
Enter a number to check if its even or odd: 97
[1] "Entered number is an odd number."
> source("~/.active-rstudio-document")
Enter a number to check if its even or odd: 25
[1] "Entered number is an odd number."
>
```

Problem Statement 12: Write a R script to find the multiplication table of inputted number.

Objective: To find the multiplication table of inputted number.

Theory:
readline() function used to read input from the user.
as.integer() function used to transform read data into an integer.
print() function to print data to console.
for() function repeats statement as long as condition is TRUE.

R Script:

```
num = as.integer(readline(prompt="Enter a number to show multiplication table of: "))

for (i in 1:10) {
  print(paste(num, "*", i, "=", num*i))
}
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number to show multiplication table of: 8
[1] "8 * 1 = 8"
[1] "8 * 2 = 16"
[1] "8 * 3 = 24"
[1] "8 * 4 = 32"
[1] "8 * 5 = 40"
[1] "8 * 6 = 48"
[1] "8 * 7 = 56"
[1] "8 * 8 = 64"
[1] "8 * 9 = 72"
[1] "8 * 10 = 80"
> source("~/.active-rstudio-document")
Enter a number to show multiplication table of: 63
[1] "63 * 1 = 63"
[1] "63 * 2 = 126"
[1] "63 * 3 = 189"
[1] "63 * 4 = 252"
[1] "63 * 5 = 315"
[1] "63 * 6 = 378"
[1] "63 * 7 = 441"
[1] "63 * 8 = 504"
[1] "63 * 9 = 567"
[1] "63 * 10 = 630"
>
```

Problem Statement 13: Write a R script to find the factorial of a number without using factorial function.

Objective: To find the factorial of a number without using factorial function.

Theory:
readline() function used to read input from the user.
as.integer() function used to transform read data into an integer..
print() function to print data to console.
for() function repeats statement as long as condition is TRUE.

R Script:

```
num = as.integer(readline(prompt="Enter a number to find factorial of: "))
fac = 1

for (i in 2:num) {
  fac = fac * i
}

print(paste("Factorial of", num, "is", fac))
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number to find factorial of: 5
[1] "Factorial of 5 is 120"
> source("~/.active-rstudio-document")
Enter a number to find factorial of: 35
[1] "Factorial of 35 is 1.03331479663861e+40"
>
```

Problem Statement 14: Write a R script to check if the input number is prime or not.

Objective: To check if the input number is prime or not.

Theory:
readline() function used to read input from the user.
as.integer() function used to transform read data into an integer..
print() function to print data to console.
for() function repeats statement as long as condition is TRUE.
sqrt() function returns the square root of the argument.

R Script:

```
num = as.integer(readline(prompt="Enter a number to check whether its prime or not: "))
prime = 1

if (num == 0 || num == 1) {
  prime = 0
} else {
  for (i in 2:sqrt(num)) {
            if (num %% i == 0) {
            prime = 0
            }
  }
}

if (prime == 1) {
  print(paste("Entered number", num, "is a prime number."))
} else {
  print(paste("Entered number", num, "is NOT a prime number."))
}
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number to check whether its prime or not: 995
[1] "Entered number 995 is NOT a prime number."
> source("~/.active-rstudio-document")
Enter a number to check whether its prime or not: 97
[1] "Entered number 97 is a prime number."
> source("~/.active-rstudio-document")
Enter a number to check whether its prime or not: 1
[1] "Entered number 1 is NOT a prime number."
> source("~/.active-rstudio-document")
Enter a number to check whether its prime or not: 0
[1] "Entered number 0 is NOT a prime number."
>
```

Problem Statement 15: Write a R script to check whether the number is palindrome or not.

Objective: To check whether the number is palindrome or not.

Theory:
readline() function used to read input from the user.
as.integer() function used to transform read data into an integer..
print() function to print data to console.
while() function repeats statement as long as the condition is TRUE.

R Script:

```
num = as.integer(readline(prompt="Enter a number to check whether its a palindrome number or not: "))
copy_num = num
rev = 0
while (copy_num != 0) {
  rem <- copy_num %% 10
  rev <- (rev * 10) + rem
  copy_num <- copy_num %/% 10
}

if (rev == num) {
  print(paste(num, "is a palindrome number."))
} else {
  print(paste(num, "is not a palindrome number."))
}
```

Output:

```
> source("~/.active-rstudio-document")
Enter a number to check whether its a palindrome number or not: 1251
1251 is not a palindrome number.
> source("~/.active-rstudio-document")
Enter a number to check whether its a palindrome number or not: 1234321
1234321 is a palindrome number.
>
```