Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
cat("------\n")

# If...Else inside While Loop (Yahtzee! Example)
dice <- 1
while (dice <= 6) {
  if (dice < 6) {
    print("No Yahtzee")
  } else {
    print("Yahtzee!")
  }
  dice <- dice + 1
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
------
[1] 1
[1] 2
[1] 3
------
[1] 1
[1] 2
[1] 4
[1] 5
```

```
[1] "No Yahtzee"
[1] "No Yahtzee"
[1] "No Yahtzee"
[1] "No Yahtzee"
[1] "No Yahtzee"
[1] "Yahtzee!"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# AND operator
a <- 200
b <- 33
c <- 500
if (a > b & c > a) {
  print("Both conditions are true")
}

# OR operator
if (a > b | a > c) {
  print("At least one of the conditions is true")
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
[1] 12
[1] TRUE
[1] TRUE
[1] FALSE
[1] "Hello World"
We are the so-called "Vikings", from the north.
[1] TRUE
[1] FALSE
[1] FALSE
[1] TRUE
```

```
[1] 5
[1] 50
[1] 2
[1] 1e+05
[1] 0
[1] 2
[1] 3
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
 [1]  1  2  3  4  5  6  7  8  9 10
[1] TRUE
[1] "b is greater than a"
[1] "a and b are equal"
[1] "a is greater than b"
[1] "b is not greater than a"
[1] "Above ten"
[1] "and also above 20!"
[1] "Both conditions are true"
[1] "At least one of the conditions is true"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
x <- 41

if (x > 10) {
  print("Above ten")
  if (x > 20) {
    print("and also above 20!")
  } else {
    print("but not above 20.")
  }
} else {
  print("below 10.")
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
[1] 12
[1] TRUE
[1] TRUE
[1] FALSE
[1] "Hello World"
We are the so-called "Vikings", from the north.
[1] TRUE
[1] FALSE
[1] FALSE
[1] TRUE
```

```
[1] 5
[1] 50
[1] 2
[1] 1e+05
[1] 0
[1] 2
[1] 3
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
 [1]  1  2  3  4  5  6  7  8  9 10
[1] TRUE
[1] "b is greater than a"
[1] "a and b are equal"
[1] "a is greater than b"
[1] "b is not greater than a"
[1] "Above ten"
[1] "and also above 20!"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# ------------------------------
# if ... else Statement
# ------------------------------
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else {
  print("b is not greater than a")
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "b is greater than a"
[1] "a and b are equal"
[1] "a is greater than b"
[1] "b is not greater than a"
```

Tweet to @rdrrHQ

GitHub issue tracker

ian@mutexlabs.com

Personal blog

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
print(seq_nums)

# %in% Operator
colors <- c("red", "green", "blue")
print("green" %in% colors)
print("yellow" %in% colors)

# Matrix Multiplication
Matrix1 <- matrix(c(1, 2, 3, 4), nrow = 2)
Matrix2 <- matrix(c(2, 0, 1, 2), nrow = 2)
result <- Matrix1 %*% Matrix2
print(result)
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 13
[1] 7
[1] 30
[1] 3.333333
[1] 1000
[1] 1
[1] 3
[1] 5
[1] 10
[1] 5
[1] 5
[1] FALSE
[1] TRUE
[1] FALSE
```

```
[1] TRUE
[1] FALSE
[1] FALSE
[1] TRUE
[1] TRUE
[1] FALSE
[1] 1 2 3 4 5
[1] TRUE
[1] FALSE
     [,1] [,2]
[1,]    2    7
[2,]    4   10
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else {
  print("b is not greater than a")
}

# Boolean with Vectors
nums <- c(1, 2, 3, 4, 5)
print(nums > 3)    # FALSE FALSE FALSE TRUE TRUE
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] TRUE
[1] FALSE
[1] FALSE
[1] TRUE
[1] "b is not greater than a"
[1] FALSE FALSE FALSE  TRUE  TRUE
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# Carriage Return
str4 <- "Hello\rWorld"  # 'World' overwrites 'Hello' in some consoles
cat(str4, "\n\n")

# Tab
str5 <- "Name:\tJohn"
cat(str5, "\n\n")

# Backspace (might not work in all environments)
str6 <- "Hello\b World"
cat(str6, "\n")
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "We are the so-called \"Vikings\", from the north."
We are the so-called "Vikings", from the north.

This is a backslash: \

First Line
Second Line

HelloWorld

Name:   John

Hello World
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# String length
print(nchar(str))

# Check for substring
print(grepl("Line", str))
print(grepl("XYZ", str))

# Combine strings
str1 <- "Hello"
str2 <- "World"
print(paste(str1, str2))
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello"
Line one
Line two
Line three[1] 28
[1] TRUE
[1] FALSE
[1] "Hello World"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
grepl("H", str)       # TRUE
grepl("Hello", str)   # TRUE
grepl("X", str)       # FALSE

# --------------------------------------
# Combine Two Strings
# --------------------------------------

str1 <- "Hello"
str2 <- "World"
print(paste(str1, str2))  # Output: "Hello World"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "hello"
[1] "hello"
[1] "Hello"
[1] "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit,\nsed do eiusmod tempor incididunt\nut la
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.[1] 12
[1] TRUE
[1] TRUE
[1] FALSE
[1] "Hello World"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
print(max(5, 10, 15))  # Output: 15
print(min(5, 10, 15))  # Output: 5

# Square Root
print(sqrt(16))        # Output: 4

# Absolute Value
print(abs(-4.7))       # Output: 4.7

# Rounding Functions
print(ceiling(1.4))    # Output: 2   (rounds up)
print(floor(1.4))      # Output: 1   (rounds down)
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 15
[1] 5
[1] 15
[1] 5
[1] 4
[1] 4.7
[1] 2
[1] 1
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
y <- 2          # numeric

# Convert from Integer to Numeric
a <- as.numeric(x)

# Convert from Numeric to Integer
b <- as.integer(y)

print(a)            # 1
print(b)            # 2
print(class(a))     # "numeric"
print(class(b))     # "integer"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 10.5
[1] 55
[1] "numeric"
[1] "numeric"
[1] 1000
[1] 55
[1] "integer"
[1] "integer"
[1] 3+5i
[1] 0+5i
[1] "complex"
[1] "complex"
[1] 1
[1] 2
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# ------------------------------
# Number Types in R
# ------------------------------

# Numeric (default type for numbers)
x <- 10.5
y <- 55

print(x)            # 10.5
print(y)            # 55
print(class(x))     # "numeric"
print(class(y))     # "numeric"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 10.5
[1] 55
[1] "numeric"
[1] "numeric"
[1] 1000
[1] 55
[1] "integer"
[1] "integer"
[1] 3+5i
[1] 0+5i
[1] "complex"
[1] "complex"
[1] 1
[1] 2
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
print(x)          # 9+3i
print(class(x))   # "complex"

# 4. Character / String
x <- "R is exciting"
print(x)          # "R is exciting"
print(class(x))   # "character"

# 5. Logical / Boolean (TRUE or FALSE)
x <- TRUE
print(x)          # TRUE
print(class(x))   # "logical"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "numeric"
[1] "character"
[1] 10.5
[1] "numeric"
[1] 1000
[1] "integer"
[1] 9+3i
[1] "complex"
[1] "R is exciting"
[1] "character"
[1] TRUE
[1] "logical"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# ---------------------------
# Case Sensitivity Example
# ---------------------------

age <- 25
Age <- 30
AGE <- 35

# All three are different
print(age)  # 25
print(Age)  # 30
print(AGE)  # 35
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "John"
[1] "John"
[1] "John"
[1] "John"
[1] "John"
[1] "John"
[1] 25
[1] 30
[1] 35
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# ---------------------------
# Assigning Same Value to Multiple Variables
# ---------------------------

# Assign the same value "Orange" to var1, var2, and var3 in one line
var1 <- var2 <- var3 <- "Orange"

# Print variable values
var1   # Output: "Orange"
var2   # Output: "Orange"
var3   # Output: "Orange"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Orange"
[1] "Orange"
[1] "Orange"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
num1 + num2

# Trying to combine string and number directly (will give error)
# num <- 5
# text <- "Some text"
# num + text   # ❌ Error: non-numeric argument

# Correct way to combine string and number
num <- 5
paste("The number is", num)
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello World!"
[1] 5
[1] 10
[1] 25
[1] 10
[1] "Hello World!"
[1] "Hello World!"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] "Hello World!"
[1] "Hello again!"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# You can also add a comment at the end of a line
"Hello World!" # This is a comment at the end

# Comments can prevent code execution
# "Good morning!"      # This line won't execute
"Good night!"          # Only this line executes

# Multiline Comments in R (no special syntax like /* */ in Java)
# This is a comment
# written in
# more than just one line
"Hello World!"         # Output still works after multiline comment
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello World!"
[1] "Hello World!"
[1] "Good night!"
[1] "Hello World!"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# Assigning values using <- (preferred way)
name <- "John"
age <- 40

# Outputting variable values without using print()
name   # Outputs: "John"
age    # Outputs: 40

# Assigning values using = (less preferred but valid)
city = "New York"
city   # Outputs: "New York"
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "John"
[1] 40
[1] "New York"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# ✅ Example where print() is REQUIRED (inside a loop or curly braces)
for (x in 1:10) {
  print(x)             # Must use print to show each value of x
}

# ✅ Another example using curly braces
{
  y <- 7 * 3
  print(paste("The value of y is", y))
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello World!"
[1] "Hello World!"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] "The value of y is 21"
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
5 + 5        # Addition
10 - 3       # Subtraction
4 * 6        # Multiplication
20 / 4       # Division

# Combining text and output using print() function
print("This is a printed message.")
print(100)

# Using cat() function to combine text and variable
a <- 50
cat("The value of a is:", a)
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello World!"
[1] "Welcome to R Programming!"
[1] 5
[1] 10
[1] 25
[1] 10
[1] 7
[1] 24
[1] 5
[1] "This is a printed message."
[1] 100
The value of a is: 50
```

myCompiler

R

R

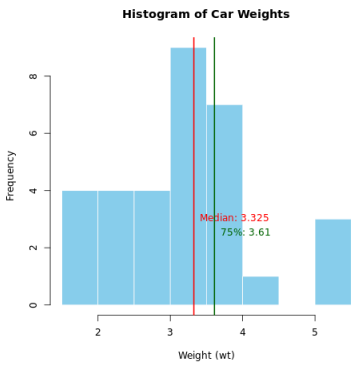
Run Save

```r
# Load mtcars dataset
Data_Cars <- mtcars

# Sorted observation of 'wt'
sorted_wt <- sort(Data_Cars$wt)
cat("Sorted 'wt' values:\n")
print(sorted_wt)

# Calculate median of 'wt'
median_wt <- median(Data_Cars$wt)
cat("\nMedian weight (wt):", median_wt, "\n")

# Calculate mode of 'wt'
mode_wt <- as.numeric(names(sort(-table(Data_Cars$wt))))
cat("Mode weight (wt):", mode_wt, "\n")

# Calculate percentiles (quartiles)
percentiles <- quantile(Data_Cars$wt)
cat("\nPercentiles (Quartiles):\n")
print(percentiles)

# Calculate specific percentile (75%)
percentile_75 <- quantile(Data_Cars$wt, 0.75)
cat("\n75th Percentile of weight (wt):", percentile_75,

# Plot histogram with median line
hist(Data_Cars$wt,
     main = "Histogram of Car Weights",
     xlab = "Weight (wt)",
     col = "skyblue",
     border = "white")

# Add median line
abline(v = median_wt, col = "red", lwd = 2)
text(median_wt, 3, labels = paste("Median:", median_wt)

# Add 75th percentile line
abline(v = percentile_75, col = "darkgreen", lwd = 2)
text(percentile_75, 2.5, labels = paste("75%:", round(p

```

Program input

Output

```
Sorted 'wt' values:
 [1] 1.513 1.615 1.835 1.93
[13] 3.150 3.170 3.190 3.21
[25] 3.730 3.780 3.840 3.84

Median weight (wt): 3.325
Mode weight (wt): 3.44

Percentiles (Quartiles):
     0%     25%     50%
1.51300 2.58125 3.32500 3.6

75th Percentile of weight (

[Execution complete with ex
```



```
41
42
```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

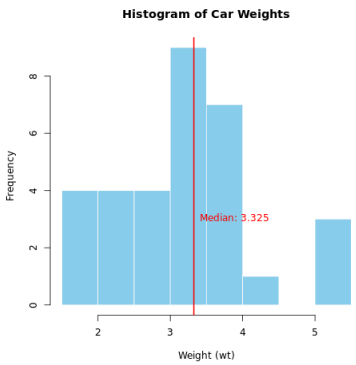
Run Save

```r
# Load the mtcars dataset
Data_Cars <- mtcars

# Calculate the median weight
median_wt <- median(Data_Cars$wt)

# Print the median
cat("Median weight (wt):", median_wt, "\n\n")

# Sort and display the 'wt' values
sorted_wt <- sort(Data_Cars$wt)
cat("Sorted 'wt' values:\n")
print(sorted_wt)

# Plot a histogram of 'wt' with the median line
hist(Data_Cars$wt,
     main = "Histogram of Car Weights",
     xlab = "Weight (wt)",
     col = "skyblue",
     border = "white")

# Add a red line for the median
abline(v = median_wt, col = "red", lwd = 2)

# Label the median on the plot
text(median_wt, 3, labels = paste("Median:", median_wt),
```

Program input

Output

```
Median weight (wt): 3.325

Sorted 'wt' values:
 [1] 1.513 1.615 1.835 1.93
[13] 3.150 3.170 3.190 3.21
[25] 3.730 3.780 3.840 3.84

[Execution complete with ex
```



Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

Run Save

```r
# Load the mtcars dataset
Data_Cars <- mtcars

# Find the maximum and minimum horsepower values
max_hp <- max(Data_Cars$hp)
min_hp <- min(Data_Cars$hp)

# Print the max and min values
cat("Maximum horsepower:", max_hp, "\n")
cat("Minimum horsepower:", min_hp, "\n")

# Find the index of the max and min horsepower
max_index <- which.max(Data_Cars$hp)
min_index <- which.min(Data_Cars$hp)

# Find the car names with max and min horsepower
car_with_max_hp <- rownames(Data_Cars)[max_index]
car_with_min_hp <- rownames(Data_Cars)[min_index]

# Print the car names
cat("Car with maximum horsepower:", car_with_max_hp, "\n"
cat("Car with minimum horsepower:", car_with_min_hp, "\n"

# Potential outliers (hypothetical examples)
cat("\nHypothetical outlier examples:\n")
cat("- A car with 11 forward gears\n")
cat("- A car with 0 horsepower\n")
cat("- A car weighing 50,000 lbs\n")
```

Program input

Output

```
Maximum horsepower: 335
Minimum horsepower: 52
Car with maximum horsepower
Car with minimum horsepower

Hypothetical outlier exampl
- A car with 11 forward gea
- A car with 0 horsepower
- A car weighing 50,000 lbs

[Execution complete with ex
```

Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

Run Save

```r
# Load the mtcars dataset
Data_Cars <- mtcars

# View the full dataset
print(Data_Cars)

# Get documentation about the mtcars dataset
?mtcars

# Get the dimensions of the dataset
dim(Data_Cars)  # Returns: 32 rows, 11 columns

# Get the names of all variables (columns)
names(Data_Cars)

# Get the row names (car names)
rownames(Data_Cars)

# Summary with improved readability using factor conversi
mtcars2 <- within(Data_Cars, {
  vs <- factor(vs, labels = c("V", "S"))
  am <- factor(am, labels = c("automatic", "manual"))
  cyl  <- ordered(cyl)
  gear <- ordered(gear)
  carb <- ordered(carb)
})

# View a statistical summary of the transformed dataset
summary(mtcars2)

# Visualize pairwise relationships
require(graphics)
pairs(Data_Cars, main = "mtcars data", gap = 1/4)

# Conditional scatter plot: mpg vs disp conditioned on cy
coplot(mpg ~ disp | as.factor(cyl), data = Data_Cars,
       panel = panel.smooth, rows = 1)

```

Program input

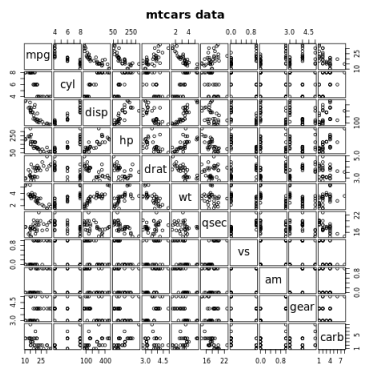
Output

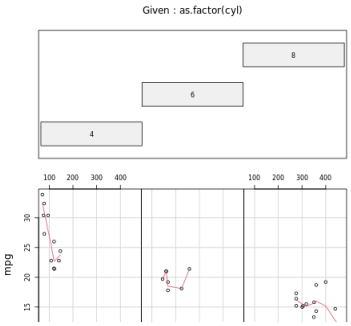
```
                     mpg
Mazda RX4           21.0
Mazda RX4 Wag       21.0
Datsun 710          22.8
Hornet 4 Drive      21.4
Hornet Sportabout   18.7
Valiant             18.1
Duster 360          14.3
Merc 240D           24.4
Merc 230            22.8
Merc 280            19.2
Merc 280C           17.8
Merc 450SE          16.4
Merc 450SL          17.3
Merc 450SLC         15.2
Cadillac Fleetwood  10.4
Lincoln Continental 10.4
Chrysler Imperial   14.7
Fiat 128            32.4
Honda Civic         30.4
Toyota Corolla      33.9
Toyota Corona       21.5
Dodge Challenger    15.5
AMC Javelin         15.2
Camaro Z28          13.3
Pontiac Firebird    19.2
Fiat X1-9           27.3
Porsche 914-2       26.0
Lotus Europa        30.4
Ford Pantera L      15.8
Ferrari Dino        19.7
Maserati Bora       15.0
```

```
Volvo 142E          21.4
mtcars                pac

_M_o_t_o_r _T_r_e

_D_e_s_c_r_i_p_t_

     The data was extract
     and comprises fuel c
     and performance for

_U_s_a_g_e:

     mtcars

_F_o_r_m_a_t:

     A data frame with 32

       [, 1]  mpg   Miles
       [, 2]  cyl   Numbe
       [, 3]  disp  Displ
       [, 4]  hp    Gross
       [, 5]  drat  Rear
       [, 6]  wt    Weigh
       [, 7]  qsec  1/4 m
       [, 8]  vs    Engin
       [, 9]  am    Trans
       [,10]  gear  Numbe
       [,11]  carb  Numbe

_N_o_t_e:

     Henderson and Vellem
     'Hocking [original t
     rotary engine as a s
     flat engine as a V e
     Mercedes 240D, have
     be made with previou

_S_o_u_r_c_e:

     Henderson and Vellem
     interactively.  _Bio

_E_x_a_m_p_l_e_s:
```

```
require(graphics)
pairs(mtcars, main =
coplot(mpg ~ disp |
       panel = panel
## possibly more mea
mtcars2 <- within(mt
   vs <- factor(vs,
   am <- factor(am,
   cyl  <- ordered(c
   gear <- ordered(g
   carb <- ordered(c
})
summary(mtcars2)
```

```
[1] 32 11
 [1] "mpg"  "cyl"  "disp"
[11] "carb"
 [1] "Mazda RX4"
 [4] "Hornet 4 Drive"
 [7] "Duster 360"
[10] "Merc 280"
[13] "Merc 450SL"
[16] "Lincoln Continental
[19] "Honda Civic"
[22] "Dodge Challenger"
[25] "Pontiac Firebird"
[28] "Lotus Europa"
[31] "Maserati Bora"
      mpg        cyl
 Min.   :10.40   4:11   M
 1st Qu.:15.43   6: 7   1
 Median :19.20   8:14   M
 Mean   :20.09          M
 3rd Qu.:22.80          3
 Max.   :33.90          M
       wt             qse
 Min.   :1.513   Min.   :
 1st Qu.:2.581   1st Qu.:
 Median :3.325   Median :
 Mean   :3.217   Mean   :
 3rd Qu.:3.610   3rd Qu.:
 Max.   :5.424   Max.   :

[Execution complete with
```





Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R Run Save

```r
# ----------------------------
# R STATISTICS BASICS
# ----------------------------

# Sample dataset
data <- c(4, 8, 6, 5, 3, 8, 9, 4, 7, 6)

# Mean (average)
mean_value <- mean(data)
cat("Mean:", mean_value, "\n")

# Median (middle value)
median_value <- median(data)
cat("Median:", median_value, "\n")

# Mode (most frequent value) - R doesn't have a built-i
get_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
mode_value <- get_mode(data)
cat("Mode:", mode_value, "\n")

# Minimum and Maximum
min_val <- min(data)
max_val <- max(data)
cat("Min:", min_val, "| Max:", max_val, "\n")

# Percentiles (25th and 75th)
percentiles <- quantile(data, probs = c(0.25, 0.75))
cat("25th Percentile:", percentiles[1], "| 75th Percent

# Variance
variance <- var(data)
cat("Variance:", variance, "\n")

# Standard Deviation
std_dev <- sd(data)
cat("Standard Deviation:", std_dev, "\n")

```

Program input

Output

```
Mean: 6
Median: 6
Mode: 4
Min: 3 | Max: 9
25th Percentile: 4.25 | 75t
Variance: 4
Standard Deviation: 2
Covariance between x and y:
Correlation between x and y

[Execution complete with ex
```

```r
# Covariance and Correlation between two variables
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)

cov_xy <- cov(x, y)
cor_xy <- cor(x, y)

cat("Covariance between x and y:", cov_xy, "\n")
cat("Correlation between x and y:", cor_xy, "\n")


```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

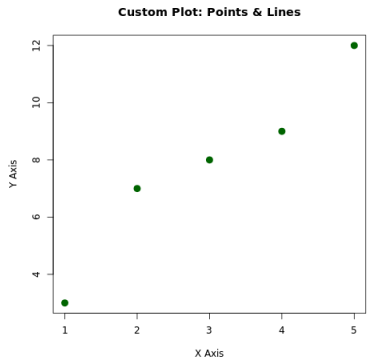
Run Save

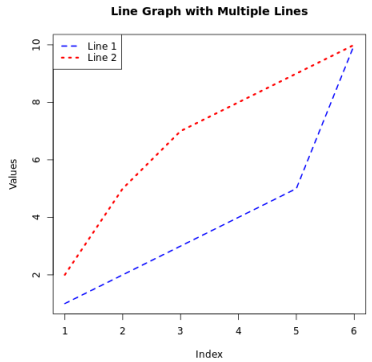
```r
# ----------------------------
# FACTORS
# ----------------------------
music_genre <- factor(
  c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz"
  levels = c("Classic", "Jazz", "Pop", "Rock", "Opera")
)

# Modify and view factor
music_genre[3] <- "Opera"
print(music_genre)
print(levels(music_genre))
print(length(music_genre))


# ----------------------------
# POINT PLOT
# ----------------------------
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)

plot(x, y,
     main = "Custom Plot: Points & Lines",
     xlab = "X Axis", ylab = "Y Axis",
     col = "darkgreen", cex = 1.5, pch = 19)


# ----------------------------
# LINE PLOTS
# ----------------------------
line1 <- c(1, 2, 3, 4, 5, 10)
line2 <- c(2, 5, 7, 8, 9, 10)

plot(line1, type = "l", col = "blue", lwd = 2, lty = 2,
     main = "Line Graph with Multiple Lines",
     xlab = "Index", ylab = "Values")

lines(line2, type = "l", col = "red", lwd = 3, lty = 3)

legend("topleft",
```

Program input

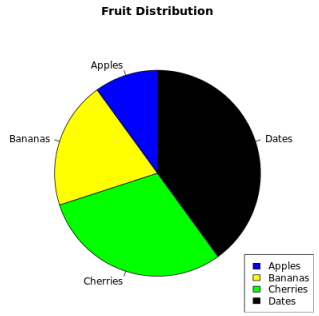
Output

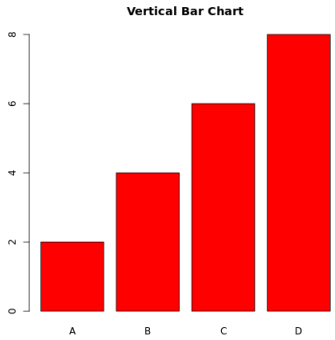
```
[1] Jazz    Rock    Opera
Levels: Classic Jazz Pop Ro
[1] "Classic" "Jazz"    "Po
[1] 8

[Execution complete with ex
```

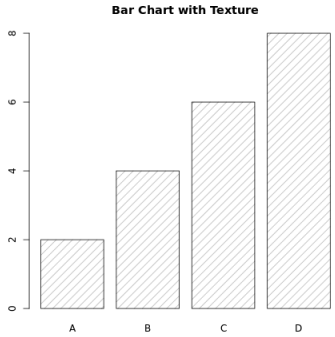




```r
        legend = c("Line 1", "Line 2"),
        col = c("blue", "red"),
        lty = c(2, 3),
        lwd = c(2, 3))


# ----------------------------
# PIE CHARTS
# ----------------------------
x_pie <- c(10, 20, 30, 40)
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")
colors <- c("blue", "yellow", "green", "black")

pie(x_pie,
    labels = mylabel,
    main = "Fruit Distribution",
    col = colors,
    init.angle = 90)

legend("bottomright", mylabel, fill = colors)


# ----------------------------
# BAR CHARTS
# ----------------------------
x_bar <- c("A", "B", "C", "D")
y_bar <- c(2, 4, 6, 8)

# Vertical bar chart with red color
barplot(y_bar, names.arg = x_bar, col = "red",
        main = "Vertical Bar Chart")

# Bar chart with texture
barplot(y_bar, names.arg = x_bar, density = 10,
        main = "Bar Chart with Texture")

# Bar chart with custom widths
barplot(y_bar, names.arg = x_bar, width = c(1,2,3,4),
        main = "Bar Chart with Varying Width")

# Horizontal bar chart
barplot(y_bar, names.arg = x_bar, horiz = TRUE,
        main = "Horizontal Bar Chart")

```







Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

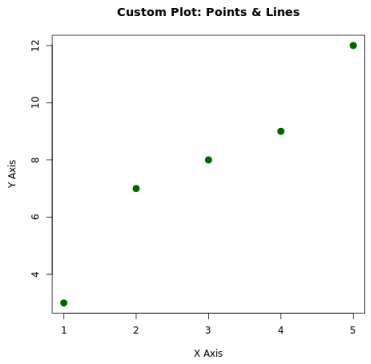
R Run Save

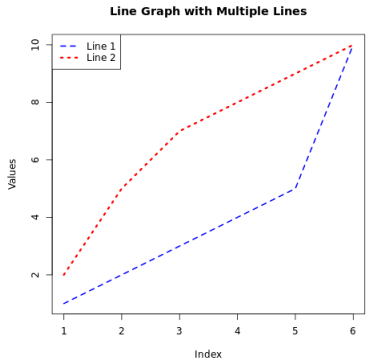
```r
# ----------------------------
# FACTORS
# ----------------------------
music_genre <- factor(
  c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz"
  levels = c("Classic", "Jazz", "Pop", "Rock", "Opera")
)

# Modify and view factor
music_genre[3] <- "Opera"
print(music_genre)
print(levels(music_genre))
print(length(music_genre))


# ----------------------------
# PLOTTING POINTS
# ----------------------------
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)

# Custom scatter plot
plot(x, y,
     main = "Custom Plot: Points & Lines",
     xlab = "X Axis", ylab = "Y Axis",
     col = "darkgreen", cex = 1.5, pch = 19)


# ----------------------------
# LINE PLOTS
# ----------------------------
line1 <- c(1, 2, 3, 4, 5, 10)
line2 <- c(2, 5, 7, 8, 9, 10)

# Base line plot
plot(line1, type = "l", col = "blue", lwd = 2, lty = 2,
     main = "Line Graph with Multiple Lines",
     xlab = "Index", ylab = "Values")

# Add second line
```

Program input

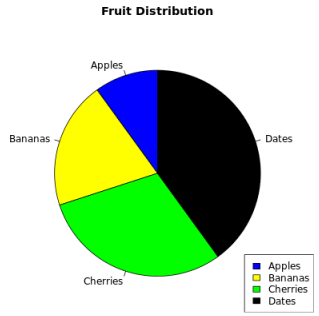
Output

```
[1] Jazz    Rock    Opera
Levels: Classic Jazz Pop Ro
[1] "Classic" "Jazz"    "Po
[1] 8

[Execution complete with ex
```





```r
lines(line2, type = "l", col = "red", lwd = 3, lty = 3)

# Add legend
legend("topleft",
       legend = c("Line 1", "Line 2"),
       col = c("blue", "red"),
       lty = c(2, 3),
       lwd = c(2, 3))


# ----------------------------
# PIE CHARTS
# ----------------------------
# Data for pie chart
x <- c(10, 20, 30, 40)
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")
colors <- c("blue", "yellow", "green", "black")

# Pie chart with labels, colors, start angle, and title
pie(x,
    labels = mylabel,
    main = "Fruit Distribution",
    col = colors,
    init.angle = 90)

# Add legend
legend("bottomright", mylabel, fill = colors)

```



Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

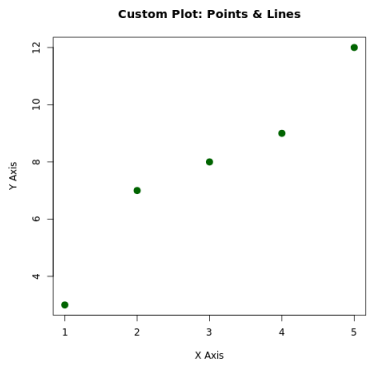
R Run Save

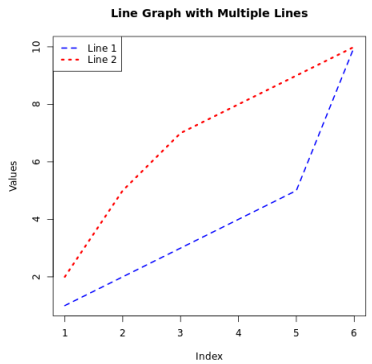
```r
# ----- FACTORS -----
# Create and customize a factor
music_genre <- factor(
  c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz"
  levels = c("Classic", "Jazz", "Pop", "Rock", "Opera")
)

# Access and modify factor
music_genre[3] <- "Opera"  # Changing value at index 3
print(music_genre)
print(levels(music_genre))
print(length(music_genre))

# ----- PLOTTING POINTS -----
# Vectors of coordinates
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)

# Scatter plot with customizations
plot(x, y,
     main = "Custom Plot: Points & Lines",
     xlab = "X Axis", ylab = "Y Axis",
     col = "darkgreen", cex = 1.5, pch = 19)

# ----- LINE PLOTS -----
# Line data
line1 <- c(1, 2, 3, 4, 5, 10)
line2 <- c(2, 5, 7, 8, 9, 10)

# Line plot with styles
plot(line1, type = "l", col = "blue", lwd = 2, lty = 2,
     main = "Line Graph with Multiple Lines",
     xlab = "Index", ylab = "Values")

# Add second line
lines(line2, type = "l", col = "red", lwd = 3, lty = 3)

# Add legend
legend("topleft",
       legend = c("Line 1", "Line 2"),
```

Program input

Output

```
[1] Jazz    Rock    Opera
Levels: Classic Jazz Pop Ro
[1] "Classic" "Jazz"    "Po
[1] 8

[Execution complete with ex
```





```
41        col = c("blue", "red"),
42        lty = c(2, 3),
43        lwd = c(2, 3))
44
```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

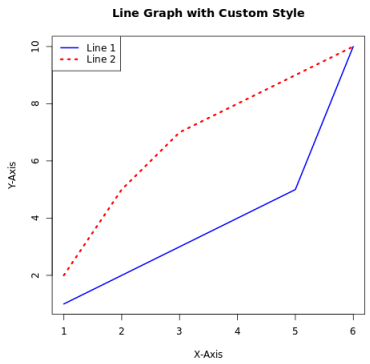
Run Save

```r
# Create sample data
line1 <- c(1, 2, 3, 4, 5, 10)
line2 <- c(2, 5, 7, 8, 9, 10)

# Basic Line Graph
plot(line1, type = "l", col = "blue", lwd = 2, lty = 1,
     main = "Line Graph with Custom Style",
     xlab = "X-Axis", ylab = "Y-Axis")

# Add another line
lines(line2, type = "l", col = "red", lwd = 3, lty = 3)

# Optional: Add legend
legend("topleft",
       legend = c("Line 1", "Line 2"),
       col = c("blue", "red"),
       lty = c(1, 3),
       lwd = c(2, 3))

```

Program input

Output

[Execution complete with ex



myCompiler

R

R

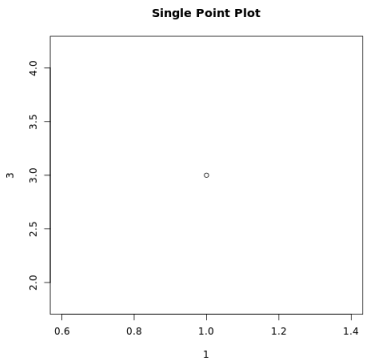
Run Save

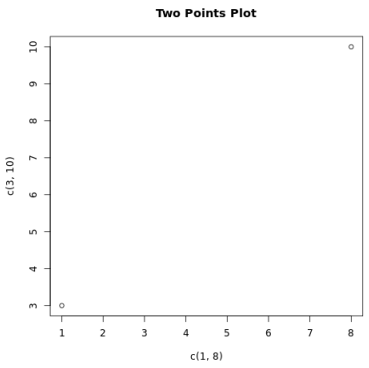
```r
# Basic Points
plot(1, 3, main="Single Point Plot")

# Multiple Points with Coordinates
plot(c(1, 8), c(3, 10), main="Two Points Plot")

# Plotting with Vectors
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)
plot(x, y, main="Multiple Points Using Vectors")

# Sequence Plot (Only x-axis provided, y = x)
plot(1:10, main="Sequence Plot")

# Line Plot
plot(1:10, type="l", main="Line Plot")

# Custom Titles and Axis Labels
plot(1:10, main="My Graph", xlab="The x-axis", ylab="The

# Colored Points
plot(1:10, col="red", main="Colored Points")

# Larger Points
plot(1:10, cex=2, main="Larger Points")

# Custom Point Shape
plot(1:10, pch=25, c
```

Program input

Output

```
Error: unexpected end of
Execution halted

[Execution complete with
```





myCompiler

R

R

Run Save

```r
#------------------------------------------------------
# R DATA FRAMES - CREATION AND OPERATIONS
#------------------------------------------------------

# Create a data frame
Data_Frame <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
print("Original Data Frame:")
print(Data_Frame)

#------------------------------------------------------
# SUMMARIZE THE DATA
#------------------------------------------------------
print("Summary of Data Frame:")
print(summary(Data_Frame))

#------------------------------------------------------
# ACCESS ITEMS
#------------------------------------------------------
print("Access column using [ ]:")
print(Data_Frame[1])

print("Access column using [[ ]] :")
print(Data_Frame[["Training"]])

print("Access column using $ :")
print(Data_Frame$Training)

#------------------------------------------------------
# ADD ROWS
#------------------------------------------------------
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110
print("Data Frame after adding a new row:")
print(New_row_DF)

```

Program input

Output

```
[1] "Original Data Frame:"
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
[1] "Summary of Data Frame:
   Training             Pul
 Length:3           Min.
 Class :character   1st Qu.
 Mode  :character   Median
                    Mean
                    3rd Qu.
                    Max.
[1] "Access column using [
  Training
1 Strength
2  Stamina
3    Other
[1] "Access column using [[
[1] "Strength" "Stamina"  "
[1] "Access column using $
[1] "Strength" "Stamina"  "
[1] "Data Frame after addin
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4 Strength   110      110
[1] "Data Frame after addin
  Training Pulse Duration S
1 Strength   100       60
2  Stamina   150       30
```

```r
#-----------------------------------------------------
# ADD COLUMNS
#-----------------------------------------------------
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000,
print("Data Frame after adding a new column:")
print(New_col_DF)

#-----------------------------------------------------
# REMOVE ROWS AND COLUMNS
#-----------------------------------------------------
Data_Frame_New <- Data_Frame[-c(1), -c(1)]
print("Data Frame after removing first row and column:
print(Data_Frame_New)

#-----------------------------------------------------
# AMOUNT OF ROWS AND COLUMNS
#-----------------------------------------------------
print("Dimensions (rows, columns):")
print(dim(Data_Frame))

print("Number of columns:")
print(ncol(Data_Frame))

print("Number of rows:")
print(nrow(Data_Frame))

#-----------------------------------------------------
# DATA FRAME LENGTH
#-----------------------------------------------------
print("Length of Data Frame (number of columns):")
print(length(Data_Frame))

#-----------------------------------------------------
# COMBINE DATA FRAMES VERTICALLY
#-----------------------------------------------------
Data_Frame1 <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame(
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
print("Combined Data Frame (Vertical):")
print(New_Data_Frame)
```

```
3    Other   120       45
[1] "Data Frame after remov
  Pulse Duration
2   150       30
3   120       45
[1] "Dimensions (rows, colu
[1] 3 3
[1] "Number of columns:"
[1] 3
[1] "Number of rows:"
[1] 3
[1] "Length of Data Frame (
[1] 3
[1] "Combined Data Frame (V
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4  Stamina   140       30
5  Stamina   150       30
6 Strength   160       20
Error: unexpected end of in
Execution halted

[Execution complete with ex
```

```r

#-----------------------------------------------------
# COMBINE DATA FRAMES HORIZONTALLY
#-----------------------------------------------------
Data_Frame3 <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame(
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3_

```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

Run Save

```r
#------------------------------------------------------
# R DATA FRAMES - CREATION AND OPERATIONS
#------------------------------------------------------

# Create a data frame
Data_Frame <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
print("Original Data Frame:")
print(Data_Frame)

#------------------------------------------------------
# SUMMARIZE THE DATA
#------------------------------------------------------
print("Summary of Data Frame:")
print(summary(Data_Frame))

#------------------------------------------------------
# ACCESS ITEMS
#------------------------------------------------------
print("Access column using [ ]:")
print(Data_Frame[1])

print("Access column using [[ ]] :")
print(Data_Frame[["Training"]])

print("Access column using $ :")
print(Data_Frame$Training)

#------------------------------------------------------
# ADD ROWS
#------------------------------------------------------
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110
print("Data Frame after adding a new row:")
print(New_row_DF)

```

Program input

Output

```
[1] "Original Data Frame:"
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
[1] "Summary of Data Frame:
   Training             Pul
 Length:3           Min.
 Class :character   1st Qu.
 Mode  :character   Median
                    Mean
                    3rd Qu.
                    Max.
[1] "Access column using [
  Training
1 Strength
2  Stamina
3    Other
[1] "Access column using [[
[1] "Strength" "Stamina"  "
[1] "Access column using $
[1] "Strength" "Stamina"  "
[1] "Data Frame after addin
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4 Strength   110      110
[1] "Data Frame after addin
  Training Pulse Duration S
1 Strength   100       60
2  Stamina   150       30
```

```r
#-----------------------------------------------------
# ADD COLUMNS
#-----------------------------------------------------
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000,
print("Data Frame after adding a new column:")
print(New_col_DF)

#-----------------------------------------------------
# REMOVE ROWS AND COLUMNS
#-----------------------------------------------------
Data_Frame_New <- Data_Frame[-c(1), -c(1)]
print("Data Frame after removing first row and column:
print(Data_Frame_New)

#-----------------------------------------------------
# AMOUNT OF ROWS AND COLUMNS
#-----------------------------------------------------
print("Dimensions (rows, columns):")
print(dim(Data_Frame))

print("Number of columns:")
print(ncol(Data_Frame))

print("Number of rows:")
print(nrow(Data_Frame))

#-----------------------------------------------------
# DATA FRAME LENGTH
#-----------------------------------------------------
print("Length of Data Frame (number of columns):")
print(length(Data_Frame))

#-----------------------------------------------------
# COMBINE DATA FRAMES VERTICALLY
#-----------------------------------------------------
Data_Frame1 <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame(
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
print("Combined Data Frame (Vertical):")
print(New_Data_Frame)
```

```
3    Other   120       45
[1] "Data Frame after remov
  Pulse Duration
2   150       30
3   120       45
[1] "Dimensions (rows, colu
[1] 3 3
[1] "Number of columns:"
[1] 3
[1] "Number of rows:"
[1] 3
[1] "Length of Data Frame (
[1] 3
[1] "Combined Data Frame (V
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4  Stamina   140       30
5  Stamina   150       30
6 Strength   160       20
Error: unexpected end of in
Execution halted

[Execution complete with ex
```

```r

#-----------------------------------------------------
# COMBINE DATA FRAMES HORIZONTALLY
#-----------------------------------------------------
Data_Frame3 <- data.frame(
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame(
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3_

```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

Run Save

```r
#----------------------------------------
# R ARRAYS - CREATION AND OPERATIONS
#----------------------------------------

# Create a 1D array (vector from 1 to 24)
thisarray <- c(1:24)
print("1D Array:")
print(thisarray)

# Create a multidimensional array: 4 rows, 3 columns, 2 m
multiarray <- array(thisarray, dim = c(4, 3, 2))
print("Multi-dimensional Array (4x3x2):")
print(multiarray)

#----------------------------------------
# ACCESS ARRAY ITEMS
#----------------------------------------

# Access a single element [row, column, matrix]
print("Element at [2, 3, 2]:")
print(multiarray[2, 3, 2])

# Access all items from the first row of matrix 1
print("All items from Row 1 of Matrix 1:")
print(multiarray[c(1), , 1])

# Access all items from the first column of matrix 1
print("All items from Column 1 of Matrix 1:")
print(multiarray[, c(1), 1])

#----------------------------------------
# CHECK IF AN ITEM EXISTS
#----------------------------------------

# Check if a specific value exists in the array
print("Is value 2 present in the array?")
print(2 %in% multiarray)

#----------------------------------------
# ARRAY DIMENSIONS AND LENGTH
```

Program input

Output

```
[1] "1D Array:"
 [1]  1  2  3  4  5  6  7
[1] "Multi-dimensional Ar
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24

[1] "Element at [2, 3, 2]
[1] 22
[1] "All items from Row 1
[1] 1 5 9
[1] "All items from Colum
[1] 1 2 3 4
[1] "Is value 2 present i
[1] TRUE
[1] "Dimensions (rows, co
[1] 4 3 2
[1] "Total number of elem
[1] 24
[1] "Looping through arra
```

```r
#----------------------------------------

# Get dimensions (rows, columns, matrices)
print("Dimensions (rows, columns, matrices):")
print(dim(multiarray))

# Get total number of elements
print("Total number of elements in array:")
print(length(multiarray))

#----------------------------------------
# LOOP THROUGH AN ARRAY
#----------------------------------------

print("Looping through array items:")
for (x in multiarray) {
  print(x)
}

```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24

[Execution complete with
```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang

SQL MySQL MongoDB Clojure D Perl

Kotlin Swift Rust Assembly

myCompiler

R

R Run Save

```r
#------------------------------
# MATRIX CREATION
#------------------------------

# Numeric matrix
thismatrix <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, nc
print("Numeric Matrix:")
print(thismatrix)

# String matrix
thismatrix <- matrix(c("apple", "banana", "cherry", "c
print("String Matrix:")
print(thismatrix)

#------------------------------
# ACCESS MATRIX ITEMS
#------------------------------

# Access single item
print("Item at [1, 2]:")
print(thismatrix[1, 2])

# Access whole row
print("Second Row:")
print(thismatrix[2, ])

# Access whole column
print("Second Column:")
print(thismatrix[, 2])

# Access more than one row
thismatrix <- matrix(c("apple", "banana", "cherry", "c
print("First and Second Rows:")
print(thismatrix[c(1, 2), ])

# Access more than one column
print("First and Second Columns:")
print(thismatrix[, c(1, 2)])

#------------------------------
```

Program input

Output

```
[1] "Numeric Matrix:"
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[1] "String Matrix:"
     [,1]     [,2]
[1,] "apple"  "cherry"
[2,] "banana" "orange"
[1] "Item at [1, 2]:"
[1] "cherry"
[1] "Second Row:"
[1] "banana" "orange"
[1] "Second Column:"
[1] "cherry" "orange"
[1] "First and Second Rows:
     [,1]     [,2]     [,3]
[1,] "apple"  "orange" "pea
[2,] "banana" "grape"  "mel
[1] "First and Second Colum
     [,1]     [,2]
[1,] "apple"  "orange"
[2,] "banana" "grape"
[3,] "cherry" "pineapple"
[1] "Matrix After Adding Co
     [,1]     [,2]        [
[1,] "apple"  "orange"    "
[2,] "banana" "grape"     "
[3,] "cherry" "pineapple" "
[1] "Matrix After Adding Ro
     [,1]         [,2]
[1,] "apple"      "orange"
```

```r
# ADD ROWS AND COLUMNS
#------------------------------

# Add column
newmatrix_col <- cbind(thismatrix, c("strawberry", "bl
print("Matrix After Adding Column:")
print(newmatrix_col)

# Add row
newmatrix_row <- rbind(thismatrix, c("strawberry", "bl
print("Matrix After Adding Row:")
print(newmatrix_row)

#------------------------------
# REMOVE ROWS AND COLUMNS
#------------------------------

thismatrix <- matrix(c("apple", "banana", "cherry", "c
thismatrix <- thismatrix[-c(1), -c(1)]
print("Matrix After Removing First Row and First Colum
print(thismatrix)

#------------------------------
# CHECK IF ITEM EXISTS
#------------------------------

thismatrix <- matrix(c("apple", "banana", "cherry", "c
print("Check if 'apple' Exists:")
print("apple" %in% thismatrix)

#------------------------------
# ROWS, COLUMNS AND LENGTH
#------------------------------

print("Matrix Dimensions (rows, columns):")
print(dim(thismatrix))

print("Total Number of Elements:")
print(length(thismatrix))

#------------------------------
# LOOP THROUGH MATRIX
#------------------------------

print("Loop Through Matrix Items:")
for (rows in 1:nrow(thismatrix)) {
  for (columns in 1:ncol(thismatrix)) {
    print(thismatrix[rows, columns])
  }
}
```

```
[2,] "banana"     "grape"
[3,] "cherry"     "pineappl
[4,] "strawberry" "blueberr
[1] "Matrix After Removing
[1] "mango"     "pineapple"
[1] "Check if 'apple' Exist
[1] TRUE
[1] "Matrix Dimensions (row
[1] 2 2
[1] "Total Number of Elemen
[1] 4
[1] "Loop Through Matrix It
[1] "apple"
[1] "cherry"
[1] "banana"
[1] "orange"
[1] "Combined Matrix (Rows)
     [,1]     [,2]
[1,] "apple"  "cherry"
[2,] "banana" "grape"
[3,] "orange" "pineapple"
[4,] "mango"  "watermelon"
[1] "Combined Matrix (Colum
     [,1]     [,2]     [,3]
[1,] "apple"  "cherry" "ora
[2,] "banana" "grape"  "man

[Execution complete with ex
```

```r

#------------------------------
# COMBINE MATRICES
#------------------------------

Matrix1 <- matrix(c("apple", "banana", "cherry", "grap
Matrix2 <- matrix(c("orange", "mango", "pineapple", "w

# Combine as rows
Matrix_Combined_Row <- rbind(Matrix1, Matrix2)
print("Combined Matrix (Rows):")
print(Matrix_Combined_Row)

# Combine as columns
Matrix_Combined_Col <- cbind(Matrix1, Matrix2)
print("Combined Matrix (Columns):")
print(Matrix_Combined_Col)


```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R Run Save

```r
# --- Creating Vectors ---

# Vector of strings
fruits <- c("banana", "apple", "orange")
print(fruits)

# Vector of numbers
numbers <- c(1, 2, 3)
print(numbers)

# Sequence of numbers using :
numbers_seq <- 1:10
print(numbers_seq)

# Decimal sequences using :
numbers1 <- 1.5:6.5
print(numbers1)

numbers2 <- 1.5:6.3
print(numbers2)

# Vector of logical values
log_values <- c(TRUE, FALSE, TRUE, FALSE)
print(log_values)


# --- Vector Length ---
print(length(fruits))


# --- Sorting Vectors ---
fruits <- c("banana", "apple", "orange", "mango", "lemo
numbers <- c(13, 3, 5, 7, 20, 2)

print(sort(fruits))   # Alphabetical
print(sort(numbers))  # Numerical


# --- Accessing Elements ---
# Access by index
```

Program input

Output

```
[1] "banana" "apple"  "oran
[1] 1 2 3
 [1]  1  2  3  4  5  6  7
[1] 1.5 2.5 3.5 4.5 5.5 6.5
[1] 1.5 2.5 3.5 4.5 5.5
[1]  TRUE FALSE  TRUE FALSE
[1] 3
[1] "apple"  "banana" "lemo
[1]  2  3  5  7 13 20
[1] "banana"
[1] "banana" "orange"
[1] "apple"  "orange" "mang
[1] "pear"   "apple"  "oran
[1] 1 1 1 2 2 2 3 3 3
[1] 1 2 3 1 2 3 1 2 3
[1] 1 1 1 1 1 2 2 3
[1]   0  20  40  60  80 100

[Execution complete with ex
```

```r
print(fruits[1])            # First element
print(fruits[c(1, 3)])      # First and third element
print(fruits[-1])           # All except first element


# --- Modifying Elements ---
fruits[1] <- "pear"         # Change banana to pear
print(fruits)


# --- Repeating Vectors ---
repeat_each <- rep(c(1, 2, 3), each = 3)
print(repeat_each)

repeat_times <- rep(c(1, 2, 3), times = 3)
print(repeat_times)

repeat_indepent <- rep(c(1, 2, 3), times = c(5, 2, 1))
print(repeat_indepent)


# --- Generating Sequences with Custom Steps ---
numbers <- seq(from = 0, to = 100, by = 20)
print(numbers)


```


Supported languages

Deno	JavaScript	NodeJS	Python	Ruby	Go
C	C++	Java	C#	TypeScript	PHP
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

myCompiler

R

R

Run Save

```r
# Example 1: Global variable used inside a function
txt <- "awesome"  # Global variable

my_function1 <- function() {
  paste("R is", txt)
}

print(my_function1())  # Output: "R is awesome"

# Example 2: Local variable with same name doesn't affe
txt <- "global variable"  # Reset global variable

my_function2 <- function() {
  txt <- "fantastic"  # Local variable
  paste("R is", txt)
}

print(my_function2())    # Output: "R is fantastic"
print(txt)               # Output: "global variable" (u

# Example 3: Creating a global variable using <<-
my_function3 <- function() {
  txt2 <<- "fantastic"  # Creates new global variable
  paste("R is", txt2)
}

print(my_function3())    # Output: "R is fantastic"
print(txt2)              # Output: "fantastic"

# Example 4: Modifying an existing global variable usin
txt <- "awesome"  # Reset global variable

my_function4 <- function() {
  txt <<- "fantastic"  # Modify existing global variabl
  paste("R is", txt)
}

print(my_function4())    # Output: "R is fantastic"
print(paste("R is", txt)) # Output: "R is fantastic"

```

Program input

Output

```
[1] "R is awesome"
[1] "R is fantastic"
[1] "global variable"
[1] "R is fantastic"
[1] "fantastic"
[1] "R is fantastic"
[1] "R is fantastic"

[Execution complete with ex
```

Supported languages

Deno	JavaScript	NodeJS	Python	R	
C	C++	Java	C#	Typ	
Bash	R	Octave (MATLAB)	Fortran	Lua	Erlang
SQL	MySQL	MongoDB	Clojure	D	Perl
Kotlin	Swift	Rust	Assembly		

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# 1. Calling a function within another function
Nested_function <- function(x, y) {
  a <- x + y
  return(a)
}

# Call: (2 + 2) + (3 + 3) = 10
print(Nested_function(Nested_function(2, 2), Nested_function(3, 3)))
# Output: 10


# 2. Defining a function inside another function
Outer_func <- function(x) {
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 10
[1] 8
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
print(my_function("Sweden"))  # Output: "I am from Sweden"
print(my_function("India"))   # Output: "I am from India"
print(my_function())          # Output: "I am from Norway"
print(my_function("USA"))     # Output: "I am from USA"

# 5. Function with Return Value
my_function <- function(x) {
  return(5 * x)
}
print(my_function(3))  # Output: 15
print(my_function(5))  # Output: 25
print(my_function(9))  # Output: 45
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "Hello World!"
[1] "Peter Griffin"
[1] "Lois Griffin"
[1] "Stewie Griffin"
[1] "Peter Griffin"
[1] "I am from Sweden"
[1] "I am from India"
[1] "I am from Norway"
[1] "I am from USA"
[1] 15
[1] 25
[1] 45
```

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
# Define two lists
adj <- list("red", "big", "tasty")
fruits <- list("apple", "banana", "cherry")

# Nested for loop
for (x in adj) {
  for (y in fruits) {
    print(paste(x, y))
  }
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] "red apple"
[1] "red banana"
[1] "red cherry"
[1] "big apple"
[1] "big banana"
[1] "big cherry"
[1] "tasty apple"
[1] "tasty banana"
[1] "tasty cherry"
```

R Package Documentation

rdrr.io home
R language documentation
Run R code online

Browse R Packages

CRAN packages
Bioconductor packages
R-Forge packages
GitHub packages

We want your feedback!

Note that we can't provide technical support on individual packages. You should contact the package authors for that.

Tweet to @rdrrHQ

GitHub issue tracker

ian@mutexlabs.com

Personal blog

Snippets

Run any R code you like. There are over *nineteen thousand* R packages preloaded.

Privacy information

Embed this on your website

List of installed packages

```
cat("------\n")

# If...Else inside For Loop: Yahtzee example
dice <- 1:6
for (x in dice) {
  if (x == 6) {
    print(paste("The dice number is", x, "Yahtzee!"))
  } else {
    print(paste("The dice number is", x, "Not Yahtzee"))
  }
}
```

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

Documentation

ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
------
[1] "apple"
[1] "banana"
[1] "cherry"
```

```
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
------
[1] "apple"
[1] "banana"
------
[1] "apple"
[1] "cherry"
------
[1] "The dice number is 1 Not Yahtzee"
[1] "The dice number is 2 Not Yahtzee"
[1] "The dice number is 3 Not Yahtzee"
[1] "The dice number is 4 Not Yahtzee"
[1] "The dice number is 5 Not Yahtzee"
[1] "The dice number is 6 Yahtzee!"
```