

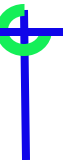


R Languages

GVGD: VÕ THỊ HỒNG TUYẾT



Nội dung

- 
-
1. Giới thiệu ngôn ngữ R
 2. Tại sao dùng R
 3. R và Google Colaboratory
 4. R basic
 5. OOP in R
 6. R data structures
 7. R graphics
 8. R statistics
 9. Data manipulation in R
 10. Machine learning in R

Giới thiệu về ngôn ngữ R

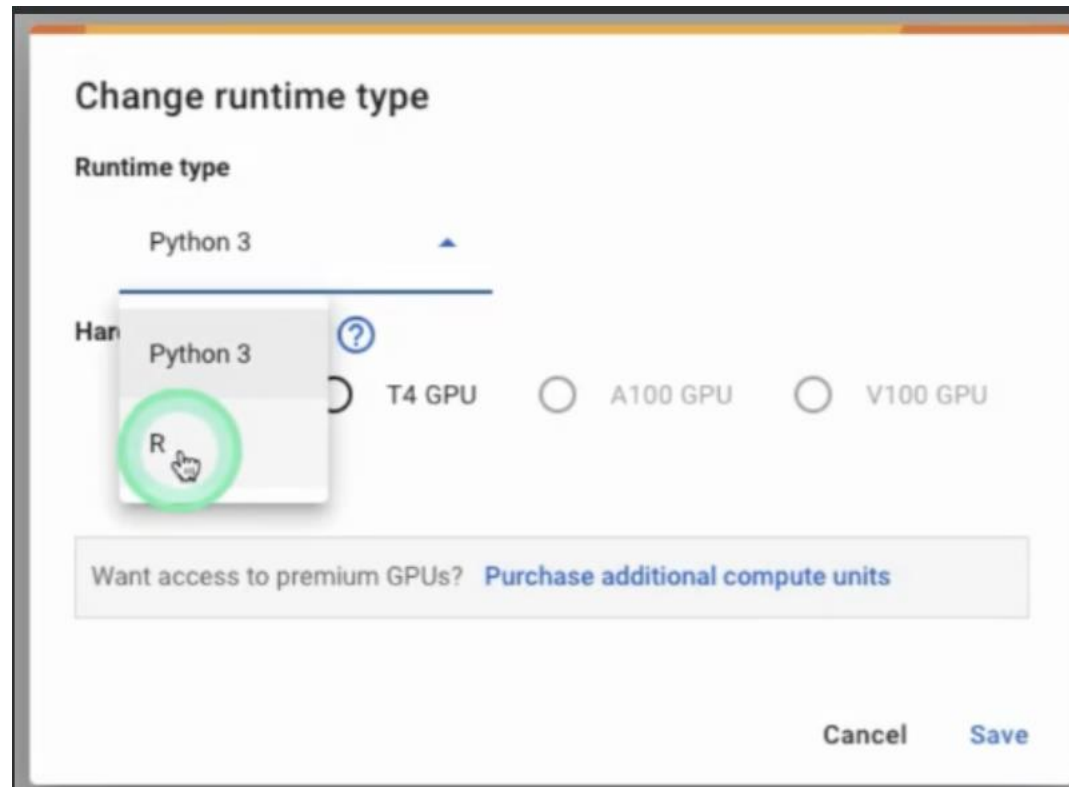
- R là ngôn ngữ lập trình thường được sử dụng trong tính toán, thống kê, trình bày đồ họa để phân tích và trực quan hóa dữ liệu.
- Cài đặt: <https://cloud.r-project.org/>
- Documents: <https://www.rdocumentation.org/>

Tại sao dùng R

- Hoạt động trên nhiều nền tảng: Windows, Linux, MacOS.
- Mã nguồn mở
- Cung cấp các kỹ thuật thống kê: kiểm tra thống kê, phân loại, phân cụm, giảm dữ liệu, ...
- Trực quan hóa dễ dàng: đồ thị, biểu đồ, ô hộp, ô phân tán, ...

R và Google Colaboratory

➤ Menu *Runtime* → *Change Runtime Type*



R basic

- Chú thích (comment): `#`
- Khai báo biến (variables): `variable_name <- value`
- Xuất (output): `print(variable_name)` hoặc `variable_name`
- Nhập (input):

`variable_name <- as.numeric(readline(prompt = "Enter a ...: "))`

- Nối chuỗi (concat):

`variable_name <- value`

`paste(variable_name, text)`

(Hoặc giữa 2 text hoặc 2 variable_name)

- Gán liên tục: `var1 <- var2 <- var3 <- value`

➤ Kiểu dữ liệu (DataType):

```
# numeric  
x <- 7.5  
class(x)
```

```
# integer  
x <- 1000L  
class(x)
```

```
# complex  
x <- 9i + 3  
class(x)
```

```
# character/string  
x <- "R language"  
class(x)
```

```
# logical/boolean  
x <- FALSE  
class(x)
```

- **numeric** - (10.5, 55, 787)
- **integer** - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- **complex** - (9 + 3i, where "i" is the imaginary part)
- **character** (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- **logical** (a.k.a. boolean) - (TRUE or FALSE)

'numeric'
'integer'
'complex'
'character'
'logical'

- **Ép kiểu (Type conversion):**
- `as.numeric()`
 - `as.integer()`
 - `as.complex()`

```
x <- 1L # integer  
y <- 2  # numeric
```

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

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

```
# print values of x and y  
x  
y
```

```
# print the class name of a and b  
class(a)  
class(b)
```

```
[1] 1  
[1] 2  
[1] "numeric"  
[1] "integer"
```


➤ Tính toán cơ bản

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
^	Exponent	$x ^ y$
%%	Modulus (Remainder from division)	$x \% \% y$
%/%	Integer Division	$x \% / \% y$

R basic

➤ Tính toán cơ bản

```
3 + 7  
8 * 2  
9 / 4  
10 - 5  
10 %% 3
```

```
10  
16  
2.25  
5  
1
```

```
max (7, -5, 10, 6)  
min (8, 5, 10)
```

```
10  
5
```

```
sqrt(16)  
abs(-6.5)  
ceiling(1.3)  
floor(1.3)
```

```
4  
6.5  
2  
1
```

➤ Biểu thức quan hệ

Operator	Name	Example
==	Equal	<code>x == y</code>
!=	Not equal	<code>x != y</code>
>	Greater than	<code>x > y</code>
<	Less than	<code>x < y</code>
>=	Greater than or equal to	<code>x >= y</code>
<=	Less than or equal to	<code>x <= y</code>

➤ And: &

➤ Or: |

R basic

➤ If...else

```
if (conditions)
{
    #statement
} [else {
    #statement
}]
```

```
if (3 > 5)
{
    "3 lon hon 5"
} else {
    "3 khong lon hon 5"
}
```

'3 khong lon hon 5'

```
if (3 > 5)
{
    "3 lon hon 5"
} else if (3 == 5) {
    "3 bang 5"
} else {
    "3 khong lon hon 5"
}
```

'3 khong lon hon 5'

R basic

break: stop loops
Next: skip an iteration

➤ While

```
while (condition) {  
  #statement  
}
```

➤ For

```
for(variable_name in list/range) {  
  #statement  
}
```

Trong đó list:

```
fruits <- list("apple", "banana", "cherry")  
  
for (x in fruits) {  
  print(x)  
}  
  
[1] "apple"  
[1] "banana"  
[1] "cherry"
```

range:

```
for (i in 1:10)  
  print(i)
```

```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

```
for (i in seq(1, 10, by = 2))  
  print(i)
```

```
[1] 1  
[1] 3  
[1] 5  
[1] 7  
[1] 9
```

R basic

➤ Function

```
function_name <- function(parameters) {  
    #statement  
    #return(expressions)  
}
```

Parameters list: ,

```
my_function <- function() {  
  print("Hello World!")  
  return (3)  
}  
my_function()
```

```
[1] "Hello World!"  
3
```

```
Nested_function <- function(x, y) {  
  a <- x + y  
  return(a)  
}  
  
Nested_function(Nested_function(2, 3), Nested_function(6, 3))
```

```
14
```

R basic

➤ Global variables

```
txt <- "awesome"  
my_function <- function() {  
  paste("R is", txt)  
}
```

```
my_function()
```

'R is awesome'

```
txt <- "global variable"  
my_function <- function() {  
  txt = "fantastic"  
  paste("R is", txt)  
}
```

```
my_function()
```

```
txt
```

'R is fantastic'

'global variable'

OOP trong R

S3	S4	R6
<ul style="list-style-type: none">- Đơn giản và phổ biến nhất trong R.- Sử dụng phương pháp class và generic.- Không kiểm tra kiểu dữ liệu chặt chẽ.	<ul style="list-style-type: none">- Có kiểm tra kiểu dữ liệu chặt chẽ.- Cho phép định nghĩa rõ ràng về class và method.- Có khả năng kế thừa và hỗ trợ để lưu trữ dữ liệu.	<ul style="list-style-type: none">- OOP hiện đại, dễ sử dụng và hiệu quả.- Hỗ trợ kế thừa, tính riêng tư và không cần định nghĩa kiểu.- Tính linh hoạt hơn trong hỗ trợ phương thức và thuộc tính.

OOP in R

➤ Class Fraction với S3

```
# Định nghĩa lớp phân số
create_fraction <- function(numerator, denominator) {
  if (denominator == 0) {
    stop("Mẫu số không được bằng 0.")
  }
  # Tạo danh sách chứa tử số và mẫu số
  fraction <- list(numerator = numerator, denominator = denominator)
  class(fraction) <- "Fraction"
  return(fraction)
}

# Hàm để đơn giản hóa phân số
simplify_fraction <- function(fraction) {
  gcd <- function(a, b) {
    while (b != 0) {
      temp <- b
      b <- a %% b
      a <- temp
    }
    return(abs(a))
  }

  divisor <- gcd(fraction$numerator, fraction$denominator)
  fraction$numerator <- fraction$numerator / divisor
  fraction$denominator <- fraction$denominator / divisor
  return(fraction)
}

# Hàm để in phân số
print.Fraction <- function(fraction) {
  cat(fraction$numerator, "/", fraction$denominator, "\n")
}

# Hàm để cộng hai phân số
add_fractions <- function(fraction1, fraction2) {
  new_numerator <- fraction1$numerator * fraction2$denominator +
    fraction2$numerator * fraction1$denominator
  new_denominator <- fraction1$denominator * fraction2$denominator
  return(simplify_fraction(create_fraction(new_numerator, new_denominator)))
}

# Ví dụ sử dụng
f1 <- create_fraction(1, 2) # Tạo phân số 1/2
f2 <- create_fraction(1, 3) # Tạo phân số 1/3

print(f1)           # In ra 1/2
print(f2)           # In ra 1/3

f3 <- add_fractions(f1, f2) # Cộng hai phân số
print(f3)           # In ra phân số kết quả
```

1 / 2
1 / 3
5 / 6

OOP in R

➤ Class Fraction với S4

```
# Định nghĩa lớp phân số
setClass("Fraction",
  slots = list(
    numerator = "numeric", # Tử số
    denominator = "numeric" # Mẫu số
  ))

# Hàm khởi tạo
setMethod("initialize", "Fraction", function(.Object, numerator, denominator) {
  if (denominator == 0) {
    stop("Mẫu số không được bằng 0.")
  }
  .Object@numerator <- numerator
  .Object@denominator <- denominator
  .Object <- simplify(.Object) # Đơn giản hóa phân số khi khởi tạo
  return(.Object)
})

# Hàm đơn giản hóa phân số
setGeneric("simplify", function(object) standardGeneric("simplify"))

setMethod("simplify", "Fraction", function(object) {
  gcd <- function(a, b) {
    while (b != 0) {
      temp <- b
      b <- a %% b
      a <- temp
    }
    return(abs(a))
  }
  divisor <- gcd(object@numerator, object@denominator)
  object@numerator <- object@numerator / divisor
  object@denominator <- object@denominator / divisor
  return(object)
})

# Hàm in phân số
setMethod("show", "Fraction", function(object) {
  cat(object@numerator, "/", object@denominator, "\n")
})

# Hàm cộng hai phân số
setGeneric("add", function(x, y) standardGeneric("add"))

setMethod("add", signature(x = "Fraction", y = "Fraction"), function(x, y) {
  new_numerator <- x@numerator * y@denominator + y@numerator * x@denominator
  new_denominator <- x@denominator * y@denominator
  return(simplify(new("Fraction", new_numerator, new_denominator)))
})

# Ví dụ sử dụng
f1 <- new("Fraction", 1, 2) # Tạo phân số 1/2
f2 <- new("Fraction", 1, 3) # Tạo phân số 1/3

print(f1) # In ra 1/2
print(f2) # In ra 1/3

f3 <- add(f1, f2) # Cộng hai phân số
print(f3) # In ra phân số kết quả
```

```
'simplify'
'add'
1 / 2
1 / 3
5 / 6
```

OOP in R

➤ Class Fraction với R6

1 / 2
1 / 3
5 / 6

```
# Tải thư viện R6
library(R6)

# Định nghĩa lớp Fraction
Fraction <- R6Class("Fraction",
  public = list(
    numerator = NULL, # Tử số
    denominator = NULL, # Mẫu số

    # Hàm khởi tạo
    initialize = function(numerator, denominator) {
      if (denominator == 0) {
        stop("Mẫu số không được bằng 0.")
      }
      self$numerator <- numerator
      self$denominator <- denominator
      self$simplify() # Đơn giản hóa phân số
    },

    # Hàm đơn giản hóa phân số
    simplify = function() {
      gcd <- function(a, b) {
        while (b != 0) {
          temp <- b
          b <- a %% b
          a <- temp
        }
        return(abs(a))
      }

      divisor <- gcd(self$numerator, self$denominator)
      self$numerator <- self$numerator / divisor
      self$denominator <- self$denominator / divisor
    },

    # Hàm in phân số
    print_fraction = function() {
      cat(self$numerator, "/", self$denominator, "\n")
    },

    # Hàm thêm hai phân số
    add = function(other) {
      new_numerator <- self$numerator * other$denominator + other$numerator * self$denominator
      new_denominator <- self$denominator * other$denominator
      return(Fraction$new(new_numerator, new_denominator))
    }
  )
)

# Ví dụ sử dụng
f1 <- Fraction$new(1, 2) # Tạo phân số 1/2
f2 <- Fraction$new(1, 3) # Tạo phân số 1/3

f1$print_fraction() # In ra 1/2
f2$print_fraction() # In ra 1/3

f3 <- f1$add(f2) # Cộng hai phân số
f3$print_fraction() # In ra phân số kết quả
```