

MAC 113 — Introdução à Ciência da Computação

Aula 11

Nelson Lago

1º/2025



Previously on MAC 113...

Indentação

```
main <- function() {  
  ...  
  while (condição) {  
    ...  
    if (condição) {  
      ...  
    } else {  
      ...  
    }  
    ...  
  }  
  ...  
}
```

- A **condição** do if/while fica entre parênteses
- O **código** a ser executado ou não fica entre chaves
- Os **parâmetros** das funções ficam entre parênteses
- O **código** das funções fica entre chaves
- Em geral, o que vai entre parênteses é “curto”, mas o que vai entre chaves pode ser bastante longo (várias linhas)
- É muito fácil para um humano se confundir com as chaves!
- A solução é a **indentação**: qualquer código entre chaves é deslocado para a direita
 - ▶ Qualquer tamanho, mas uma boa opção para **R** são 4 espaços

Cuidado com o amigo da onça!

- Talvez você tenha percebido que às vezes é possível omitir `return()` ...
- Talvez você tenha percebido que às vezes é possível omitir as chaves com `if / else / while` ...
- Talvez você tenha percebido que às vezes é possível omitir o `"\n"` no final do comando `cat()` ...

Cuidado!

Não caia em tentação!

Quando você menos esperar, isso vai causar problemas!

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia
- Portanto...
- Elas precisam ser capazes de funcionar de maneira independente do contexto

**Cada um com seu cada um e ninguém se mete
no cada um dos outros**

Nomes e memória

- Em R, só existe quem tem nome:

```
saudação <- "Olá!"  
saudação <- "Oi!"
```

- O texto **"Olá!"** não existe mais em nenhum lugar da memória após a segunda linha ser executada!

```
x <- 1  
x <- x + 1
```

- O número **1** não existe mais em nenhum lugar da memória após a segunda linha ser executada!

Nomes e memória

- Em R, só existe quem tem nome:

```
x <- 1  
y <- x  
x <- x + 1
```

- O número 1 *continua existindo* na memória após a terceira linha ser executada
 - ▶ Mas o que acontece na segunda linha?!?!?
 - » Temos duas “cópias” do número 1 na memória?
- DEPENDE
 - ▶ Em R, sim!
 - ▶ Na terceira linha, voltamos a ter apenas uma cópia
 - » (R na verdade evita copiar quando não é necessário, mas do ponto de vista do programador é “como se” sempre houvesse a cópia)

O que são “variáveis”?

- Nomes que usamos para acessar dados armazenados em algum lugar na memória do computador?
- Conceitos abstratos que representam uma informação relevante para a computação que estamos realizando?

Ambos

- Mas cada linguagem dá mais ênfase a este ou àquele ponto de vista

Variáveis e memória

- **Em python**

- ▶ $x = x + 1$ não altera o número 1 armazenado na memória, mas sim o valor associado à variável x (o número 2 “nasce” e recebe o nome x , enquanto o número 1 é “jogado fora”)
 - » *Pensando em variáveis como conceitos, isso faz sentido: o valor da variável é o conceito, e ele foi modificado; o número armazenado é **outro** número*

- **Em outras linguagens, como R e C, as coisas são diferentes:**

- ▶ $x \leftarrow y$ guarda duas “cópias” do mesmo número em lugares diferentes da memória
- ▶ $x \leftarrow x + 1$ procura o lugar na memória em que a variável x está armazenada e altera o valor que está ali
 - » *Pensando em variáveis como nomes para um dado na memória, isso faz sentido: o conteúdo daquele espaço de memória foi modificado*

E por que eu preciso me preocupar com isso?

```
metade <- function(x) {  
  x <- x / 2  
  return (x)  
}  
x <- 10  
cat(metade(x), "é a metade de", x, "\n")
```

- Na chamada a **metade()**, o número **10**, que está armazenado em **val**, é **copiado** para o nome **x**
 - ▶ O nome **x** só existe **dentro** da função **metade()**!
 - ▶ Quando a função termina, o nome **x** e seu valor deixam de existir
 - ▶ Quando o valor de **x** é modificado dentro da função, isso não altera nenhuma variável **fora** da função
 - » *E isso continua valendo mesmo que, “por coincidência”, o nome “de fora” e o nome “de dentro” sejam iguais*

Funções – escopo

- Funções são **nomes** dados a trechos de código que representam uma **ideia** bem definida e auto-contida
- Uma das vantagens é que elas podem ser usadas em contextos diferentes para realizar a computação correspondente àquela ideia
- Portanto...
- Elas precisam ser capazes de funcionar de maneira independente do contexto

**Cada um com seu cada um e ninguém se mete
no cada um dos outros**

Funções

- Não confunda **cat()** e **return**!

- ▶ **cat()** e **readline()** → comunicação com o “mundo”
- ▶ **return** e os parâmetros das funções → comunicação entre partes diferentes do programa

```
metade <- function(x) {  
  x <- x / 2  
  return (x)  
}  
val <- 10  
cat(metade(val), "é a metade de", val, "\n")
```

```
metade <- function(x) {  
  y <- x / 2  
  cat(y, "é a metade de", x, "\n")  
}  
metade(10)
```

Exercício — sistema de *login*

Imagine um sistema de *login* com três usuários:

- **Alan Turing** — UID **turing**, senha **tmachine**
- **Ada Lovelace** — UID **llace**, senha **anengine**
- **Grace Hopper** — UID **hopper**, senha **business**

Crie um sistema que lê o UID (*login*) e a senha do usuário e, se os dados estiverem corretos, escreve “Bem-vindo, [nome]!”; caso contrário, o sistema escreve “Login ou senha incorreto”.

Exercício — sistema de *login*

```
library(glue)
```

Exercício — sistema de *login*

```
library(glue)
uid <- readline("username: ")
senha <- readline("senha: ")
```

Exercício — sistema de *login*

```
library(glue)
uid <- readline("username: ")
senha <- readline("senha: ")

if (nome != "") {
  cat(glue("Bem-vindo, {nome}!"), "\n")
} else {
  cat("Login ou senha incorreto", "\n")
}
```

Exercício — sistema de *login*

```
library(glue)
uid <- readline("username: ")
senha <- readline("senha: ")
nome <- ""

if (nome != "") {
  cat(glue("Bem-vindo, {nome}!"), "\n")
} else {
  cat("Login ou senha incorreto", "\n")
}
```

Exercício — sistema de *login*

```
library(glue)
uid <- readline("username: ")
senha <- readline("senha: ")
nome <- ""
if (uid == "turing" && senha == "tmachine")
  { nome <- "Alan Turing" }

if (nome != "") {
  cat(glue("Bem-vindo, {nome}!"), "\n")
} else {
  cat("Login ou senha incorreto", "\n")
}
```

Exercício — sistema de *login*

```
library(glue)
uid <- readline("username: ")
senha <- readline("senha: ")
nome <- ""
if (uid == "turing" && senha == "tmachine")
  { nome <- "Alan Turing" }
if (uid == "llace" && senha == "anengine")
  { nome <- "Ada Lovelace" }
if (uid == "hopper" && senha == "business")
  { nome <- "Grace Hopper" }
if (nome != "") {
  cat(glue("Bem-vindo, {nome}!"), "\n")
} else {
  cat("Login ou senha incorreto", "\n")
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(glue("Bem-vindo, {nome}!"), "\n")  
  }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(glue("Bem-vindo, {nome}!"), "\n")  
  }  
}  
  
checa_login <- function(uid, senha) {  
  nome <- ""  
  if (uid == "turing" && senha == "tmachine")  
    { nome <- "Alan Turing" }  
  if (uid == "llace" && senha == "anengine")  
    { nome <- "Ada Lovelace" }  
  if (uid == "hopper" && senha == "business")  
    { nome <- "Grace Hopper" }  
  return (nome)  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(glue("Bem-vindo, {nome}!"), "\n")  
  }  
}  
  
checa_login <- function(uid, senha) {  
  name <- ""  
  if (uid == "turing" && senha == "tmachine")  
    { name <- "Alan Turing" }  
  if (uid == "llace" && senha == "anengine")  
    { name <- "Ada Lovelace" }  
  if (uid == "hopper" && senha == "business")  
    { name <- "Grace Hopper" }  
  return (name)  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(glue("Bem-vindo, {nome}!"), "\n")  
  }  
}  
  
checa_login <- function(uid, senha) {  
  
  if (uid == "turing" && senha == "tmachine")  
    { return("Alan Turing") }  
  if (uid == "llace" && senha == "anengine")  
    { return("Ada Lovelace") }  
  if (uid == "hopper" && senha == "business")  
    { return("Grace Hopper") }  
  return ("")  
}
```

Exercício — sistema de *login*

Modifique o código anterior para que o programa faça saudações diferentes em função do horário:

- “Bom dia, [nome]!” (das 6h às 12h59)
- “Boa tarde, [nome]!” (das 13h às 18h59)
- “Boa noite, [nome]!” (após as 19h ou antes das 6h)

Para saber o horário atual:

```
hora <- as.integer(format(Sys.time(), format="%H"))
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(glue("Bem-vindo, {nome}!"), "\n")  
  }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(saudação(nome), "\n")  
  }  
}
```

Exercício — sistema de *login*



Exercício — sistema de *login*

```
saudação <- function(nome) {
```

```
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  
  return(glue("Bom dia, {nome}!"))  
  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  
  return(glue("Bom dia, {nome}!"))  
  
  return(glue("Boa tarde, {nome}!"))  
  
  return(glue("Boa noite, {nome}!"))  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if ( )  
    { return(glue("Bom dia, {nome}!")) }  
  if ( )  
    { return(glue("Boa tarde, {nome}!")) }  
  if ( )  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6  
      )  
    { return(glue("Bom dia, {nome}!")) }  
  if (  
      )  
    { return(glue("Boa tarde, {nome}!")) }  
  if (  
      )  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (  
    )  
    { return(glue("Boa tarde, {nome}!")) }  
  if (  
    )  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19  
      )  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 ||      )  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  if (nome == "") {  
    cat("Login ou senha incorreto", "\n")  
  } else {  
    cat(saudação(nome), "\n")  
  }  
}  
  
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  cat(saudação(nome), "\n")  
  
}  
saudação <- function(nome) {  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  cat(saudação(nome), "\n")  
}  
saudação <- function(nome) {  
  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  cat(saudação(nome), "\n")  
}  
saudação <- function(nome) {  
  if (nome == "")  
  {  
    }  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Exercício — sistema de *login*

```
main <- function() {  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha)  
  cat(saudação(nome), "\n")  
}  
saudação <- function(nome) {  
  if (nome == "")  
    { return("Login ou senha incorreto") }  
  hora <- as.integer(format(Sys.time(), format="%H"))  
  if (hora >= 6 && hora < 13)  
    { return(glue("Bom dia, {nome}!")) }  
  if (hora >= 13 && hora < 19)  
    { return(glue("Boa tarde, {nome}!")) }  
  if (hora >= 19 || hora < 6)  
    { return(glue("Boa noite, {nome}!")) }  
}
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {
```

}

```
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  
  
  
  
  
  
  
  
  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
  if (      )  
  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
  if ( ) {  
    cat("O ponto está dentro do círculo", "\n")  
  
  }  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
  if ( ) {  
    cat("O ponto está dentro do círculo", "\n")  
  } else {  
    cat("O ponto está fora do círculo", "\n")  
  }  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
  if (distancia <= 1) {  
    cat("O ponto está dentro do círculo", "\n")  
  } else {  
    cat("O ponto está fora do círculo", "\n")  
  }  
}  
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  distancia <- sqrt(x^2 + y^2)  
  if (distancia <= 1) {  
    cat("O ponto está dentro do círculo", "\n")  
  } else {  
    cat("O ponto está fora do círculo", "\n")  
  }  
}
```

```
main()
```

Pré-exercício

Dada uma coordenada x, y , diga se o ponto correspondente está dentro do círculo de raio 1 com centro na origem.

```
main <- function() {  
  x <- as.numeric(readline("Informe x: "))  
  y <- as.numeric(readline("Informe y: "))  
  
  if (x^2 + y^2 <= 1) {  
    cat("O ponto está dentro do círculo", "\n")  
  } else {  
    cat("O ponto está fora do círculo", "\n")  
  }  
}  
main()
```

- Alguns problemas são difíceis de resolver de maneira determinística, mas resultados aproximados podem ser aceitáveis

Método de Monte Carlo

- Alguns problemas são difíceis de resolver de maneira determinística, mas resultados aproximados podem ser aceitáveis
- Uma maneira de abordar alguns desses problemas é o método de Monte Carlo: **chutar** 😂

Método de Monte Carlo

- Alguns problemas são difíceis de resolver de maneira determinística, mas resultados aproximados podem ser aceitáveis
- Uma maneira de abordar alguns desses problemas é o método de Monte Carlo: **chutar** 😂
 - ▶ Chuta valores de entrada e calcula os resultados correspondentes

Método de Monte Carlo

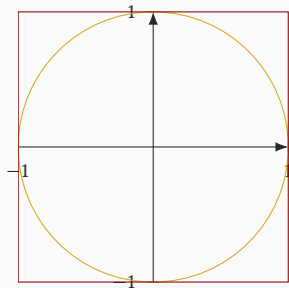
- Alguns problemas são difíceis de resolver de maneira determinística, mas resultados aproximados podem ser aceitáveis
- Uma maneira de abordar alguns desses problemas é o método de Monte Carlo: **chutar** 😂
 - ▶ Chuta valores de entrada e calcula os resultados correspondentes
 - ▶ Faz análise estatística dos resultados

Método de Monte Carlo

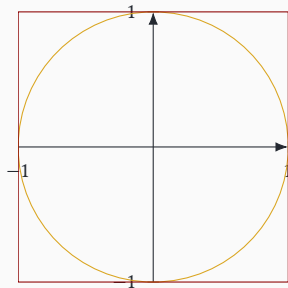
- Alguns problemas são difíceis de resolver de maneira determinística, mas resultados aproximados podem ser aceitáveis
- Uma maneira de abordar alguns desses problemas é o método de Monte Carlo: **chutar** 😂
 - ▶ Chuta valores de entrada e calcula os resultados correspondentes
 - ▶ Faz análise estatística dos resultados

```
cat(runif(1), "\n") # Número aleatório no intervalo [0,1]
```

Calculando π pelo método de Monte Carlo



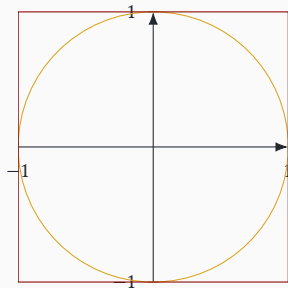
Calculando π pelo método de Monte Carlo



área do quadrado = $2 \cdot 2 = 4$

área do círculo = $\pi \cdot 1^2 = \pi$

Calculando π pelo método de Monte Carlo

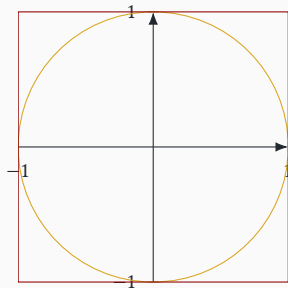


$$\text{área do quadrado} = 2 \cdot 2 = 4$$

$$\text{área do círculo} = \pi \cdot 1^2 = \pi$$

- Sorteando um ponto dentro do retângulo, a probabilidade de ele estar dentro do círculo é $\frac{\pi}{4}$

Calculando π pelo método de Monte Carlo

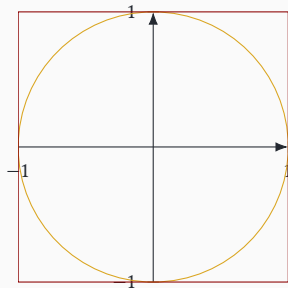


$$\text{área do quadrado} = 2 \cdot 2 = 4$$

$$\text{área do círculo} = \pi \cdot 1^2 = \pi$$

- Sorteando um ponto dentro do retângulo, a probabilidade de ele estar dentro do círculo é $\frac{\pi}{4}$
- Se sortearmos vários pontos, podemos estimar o valor de π com base na proporção de pontos dentro do círculo

Calculando π pelo método de Monte Carlo

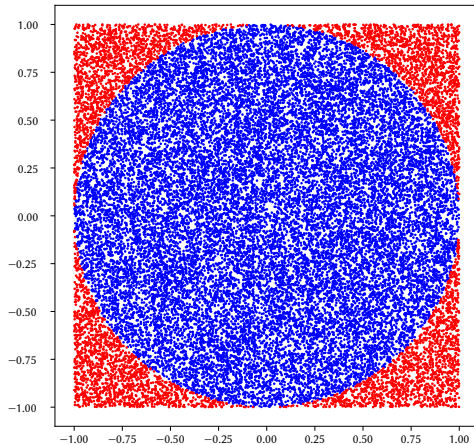


$$\text{área do quadrado} = 2 \cdot 2 = 4$$

$$\text{área do círculo} = \pi \cdot 1^2 = \pi$$

- Sorteando um ponto dentro do retângulo, a probabilidade de ele estar dentro do círculo é $\frac{\pi}{4}$
- Se sortearmos vários pontos, podemos estimar o valor de π com base na proporção de pontos dentro do círculo
 - ▶ $\pi = 4 \cdot \text{probabilidade}$

Calculando π pelo método de Monte Carlo



Calculando π pelo método de Monte Carlo



Calculando π pelo método de Monte Carlo

```
main <- function() {
```

```
}
```

```
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  i <- 1  
  while (i <= chutes) {  
  
  
  
  
  
  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  i <- 1  
  while (i <= chutes) {  
  
    i <- i + 1  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  i <- 1  
  while (i <= chutes) {  
    x <- runif(1, -1, 1)  
    y <- runif(1, -1, 1)  
  
    i <- i + 1  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  i <- 1  
  while (i <= chutes) {  
    x <- runif(1, -1, 1)  
    y <- runif(1, -1, 1)  
    if (x^2 + y^2 <= 1) { # dentro do círculo  
  
    }  
    i <- i + 1  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  
  i <- 1  
  while (i <= chutes) {  
    x <- runif(1, -1, 1)  
    y <- runif(1, -1, 1)  
    if (x^2 + y^2 <= 1) { # dentro do círculo  
      dentro <- dentro + 1  
    }  
    i <- i + 1  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Calculando π pelo método de Monte Carlo

```
main <- function() {  
  chutes <- as.integer(readline("Quantas amostras? "))  
  dentro <- 0  
  i <- 1  
  while (i <= chutes) {  
    x <- runif(1, -1, 1)  
    y <- runif(1, -1, 1)  
    if (x^2 + y^2 <= 1) { # dentro do círculo  
      dentro <- dentro + 1  
    }  
    i <- i + 1  
  }  
  cat("Pi é aproximadamente", 4*dentro/chutes, "\n")  
}  
main()
```

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: a: x:

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 25 a: x:

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 25 a: 3 x:

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 25 a: 3 x: 5

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 20 a: 3 x: 5

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 20 a: 23 x: 5

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 20 a: 23 x: 5

saída: 35

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 20 a: 23 x: 2

saída: 35

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 15 a: 23 x: 2

saída: 35

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 15 a: 38 x: 2

saída: 35

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 15 a: 38 x: 2

saída: 35

saída: 14

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 15 a: 38 x: 3

saída: 35

saída: 14

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 10 a: 38 x: 3

saída: 35

saída: 14

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 10 a: 48 x: 3

saída: 35

saída: 14

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 10 a: 48 x: 3

saída: 35

saída: 14

saída: 21

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 10 a: 48 x: 6

saída: 35

saída: 14

saída: 21

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 5 a: 48 x: 6

saída: 35

saída: 14

saída: 21

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 5 a: 53 x: 6

saída: 35

saída: 14

saída: 21

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 5 a: 53 x: 6

saída: 35

saída: 14

saída: 21

saída: 42

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 5 a: 53 x: 4

saída: 35

saída: 14

saída: 21

saída: 42

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 0 a: 53 x: 4

saída: 35

saída: 14

saída: 21

saída: 42

Exercício — pensando como o computador

O que este programa imprime se o usuário escolhe o número 5?

```
main <- function() {  
  k <- 25  
  a <- 3  
  x <- as.integer(readline("Digite um inteiro positivo: "))  
  i <- 1  
  while (i <= 5) {  
    k <- k - 5  
    a <- a + k  
    cat(x * 7, "\n")  
    x <- a %% 7  
    i <- i + 1  
  }  
}  
main()
```

k: 0 a: 53 x: 4

saída: 35

saída: 14

saída: 21

saída: 42

saída: 28

And now for something completely different

Coleções

- Lista de clientes

- **Lista de clientes**

- ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

- **Lista de clientes**

- ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

- **Lista dos divisores de um número**

- **Lista de clientes**
 - ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **Lista com o PIB anual de um país**

- **Lista de clientes**
 - ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **Lista com o PIB anual de um país**
- ...

- **Lista de clientes**

- ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

- **Lista dos divisores de um número**

- **Lista com o PIB anual de um país**

- ...

Faz sentido inventar nomes (variáveis) para cada um deles?

- **Lista de clientes**

- ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

- **Lista dos divisores de um número**

- **Lista com o PIB anual de um país**

- ...

Faz sentido inventar nomes (variáveis) para cada um deles?

- **aluno1, aluno2, aluno3...**

- **Lista de clientes**

- ▶ E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

- **Lista dos divisores de um número**

- **Lista com o PIB anual de um país**

- ...

Faz sentido inventar nomes (variáveis) para cada um deles?

- **aluno1, aluno2, aluno3...**

dica: não 😂

O que faz sentido é tratá-los como um *conjunto* que tem um nome

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- Podemos manipular o conjunto como um todo (“alunos”)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- Podemos manipular o conjunto como um todo (“`alunos`”)
- Podemos manipular um elemento específico (“`alunos[4]`”)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- Podemos manipular o conjunto como um todo (“`alunos`”)
- Podemos manipular um elemento específico (“`alunos[4]`”)
- Podemos manipular subconjuntos (“`alunos[2:7]`”)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- Podemos manipular o conjunto como um todo (“`alunos`”)
- Podemos manipular um elemento específico (“`alunos[4]`”)
- Podemos manipular subconjuntos (“`alunos[2:7]`”)
- Podemos manipular elementos um por um
(“`for (aluno in alunos) { ... }`”)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- Podemos manipular o conjunto como um todo (“alunos”)
- Podemos manipular um elemento específico (“alunos[4]”)
- Podemos manipular subconjuntos (“alunos[2:7]”)
- Podemos manipular elementos um por um
(“for (aluno in alunos) {...}”)
- ...

Coleções

Há três tipos principais de coleções:

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**
 - Nomes de pacientes

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários

- ▶ PIB anual

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço,
estado civil, CPF...

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço,
estado civil, CPF...

Pessoa: nome, colesterol,
cor do cabelo...

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço, estado civil, CPF...

Pessoa: nome, colesterol, cor do cabelo...

País: capital, PIB per capita, área cultivável...

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço, estado civil, CPF...

Pessoa: nome, colesterol, cor do cabelo...

País: capital, PIB per capita, área cultivável...

Personagem: inteligência, força, destreza...

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço, estado civil, CPF...

Pessoa: nome, colesterol, cor do cabelo...

País: capital, PIB per capita, área cultivável...

Personagem: inteligência, força, destreza...

- **Tabelas (mistura dos dois)**

Coleções

Há três tipos principais de coleções:

- **Listas de coisas “iguais”**

- ▶ Nomes de pacientes
- ▶ Salários de funcionários
- ▶ PIB anual
- ▶ Objetos para comprar

- **Composição de dados formando uma abstração**

Pessoa: nome, endereço, estado civil, CPF...

Pessoa: nome, colesterol, cor do cabelo...

País: capital, PIB per capita, área cultivável...

Personagem: inteligência, força, destreza...

- **Tabelas (mistura dos dois)**

Coleções

A primeira coleção que vamos ver é o **vector**:

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")
```

vermelho

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")  
cat(cores[3], "\n")
```

vermelho

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")  
cat(cores[3], "\n")
```

```
vermelho  
amarelo
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c('vermelho', 'azul', 'amarelo')
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")  
cat(cores[3], "\n")
```

```
vermelho  
amarelo
```

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")  
cat(cores[3], "\n")
```

```
vermelho  
amarelo
```

- Todos os elementos de um vector precisam ser do mesmo tipo

Coleções

A primeira coleção que vamos ver é o **vector**:

```
cores <- c("vermelho", "azul", "amarelo")
```

Um vector é um conjunto ordenado:

```
cat(cores[1], "\n")  
cat(cores[3], "\n")
```

```
vermelho  
amarelo
```

- **Todos os elementos de um vector precisam ser do mesmo tipo**
 - ▶ Usamos vectors para representar coleções de coisas “iguais”

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
```

```
}
main()
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
  turtle_init(300,300)
```

```
}
main()
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
  turtle_init(300,300)

  i <- 1
  while (i <= 4) {

    i <- i + 1
  }
}
main()
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
  turtle_init(300,300)

  i <- 1
  while (i <= 4) {

    turtle_forward(100)
    turtle_right(90)
    i <- i + 1
  }
}
main()
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
  turtle_init(300,300)
  cores <- c("red", "green", "blue", "yellow")
  i <- 1
  while (i <= 4) {

    turtle_forward(100)
    turtle_right(90)
    i <- i + 1
  }
}
main()
```

Vectors

```
install.packages("TurtleGraphics")
library("TurtleGraphics")
fakedraw <- function() {invisible(NULL)}
environment(fakedraw) <- asNamespace("TurtleGraphics")
assignInNamespace(".turtle_draw", fakedraw, ns="TurtleGraphics")
assignInNamespace(".turtle_undraw", fakedraw, ns="TurtleGraphics")
```

```
main <- function() {
  turtle_init(300,300)
  cores <- c("red", "green", "blue", "yellow")
  i <- 1
  while (i <= 4) {
    turtle_col(cores[i])
    turtle_forward(100)
    turtle_right(90)
    i <- i + 1
  }
}
main()
```

Repetições com coleções

```
main <- function() {
```

}

```
main()
```

Repetições com coleções

```
main <- function() {  
  primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)  
  
}  
main()
```

Repetições com coleções

```
main <- function() {  
  primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)  
  i <- 1  
  while (i <= length(primos)) {  
    i <- i + 1  
  }  
}
```

`main()`

Repetições com coleções

```
main <- function() {  
  primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)  
  i <- 1  
  while (i <= length(primos)) {  
    cat("O número", primos[i], "é primo", "\n")  
    i <- i + 1  
  }  
}  
main()
```

Repetições com coleções

```
main <- function() {  
  primos <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)  
  i <- 1  
  while (i <= length(primos)) {  
    cat("O número", primos[i], "é primo", "\n")  
    i <- i + 1  
  }  
}  
main()
```

Num show, a lista de músicas que a banda vai executar é o *setlist*. Os músicos precisam ter uma cópia impressa do *setlist* para não perderem tempo entre uma música e outra.

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
gera_setlist <- function() {  
  
  
  
  
  
  
  
  
  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
gera_setlist <- function() {  
  
  i <- 1  
  while (i <= n) {  
  
    i <- i + 1  
  }  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
gera_setlist <- function() {  
  n <- as.integer(readline("Quantas músicas? "))  
  
  i <- 1  
  while (i <= n) {  
  
    i <- i + 1  
  }  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
  
gera_setlist <- function() {  
  n <- as.integer(readline("Quantas músicas? "))  
  
  i <- 1  
  while (i <= n) {  
    música <- readline("Qual a próxima música? ")  
  
    i <- i + 1  
  }  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
gera_setlist <- function() {  
  n <- as.integer(readline("Quantas músicas? "))  
  
  i <- 1  
  while (i <= n) {  
    música <- readline("Qual a próxima música? ")  
    setlist[i] <- música  
    i <- i + 1  
  }  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
  
gera_setlist <- function() {  
  n <- as.integer(readline("Quantas músicas? "))  
  setlist <- character(n)  
  i <- 1  
  while (i <= n) {  
    música <- readline("Qual a próxima música? ")  
    setlist[i] <- música  
    i <- i + 1  
  }  
  return(setlist)  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
imprime <- function(l) {  
  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
imprime <- function(l) {  
  cat('setlist - Show da banda "Os Errantes"', "\n")  
  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
imprime <- function(l) {  
  cat('setlist - Show da banda "Os Errantes"', "\n")  
  cat("\n")  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
imprime <- function(l) {  
  cat('setlist - Show da banda "Os Errantes"', "\n")  
  cat("\n")  
  i <- 1  
  while (i <= length(l)) {  
    i <- i + 1  
  }  
}
```

Coleções

Escreva um programa que pergunta quantas músicas serão executadas, pergunta o nome de cada uma delas e imprime cinco cópias do *setlist*.

```
main <- function() {  
  sl <- gera_setlist()  
  i <- 1  
  while (i <= 5) {  
    imprime(sl)  
    i <- i + 1  
  }  
}  
  
imprime <- function(l) {  
  cat('setlist - Show da banda "Os Errantes"', "\n")  
  cat("\n")  
  i <- 1  
  while (i <= length(l)) {  
    cat(i, " - ", l[i], "\n")  
    i <- i + 1  
  }  
}
```

Dada uma lista de inteiros, imprima o maior valor da lista

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {
```

```
}
```

```
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  
}
```

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  
  maior <- lista[1]  
  for (i in 2:length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  
  i <- 1  
  while (i <= length(lista)) {  
  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
  
      }  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior valor da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior *e o menor* valores da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Dada uma lista de inteiros, imprima o maior *e o menor* valores da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "\n")  
}  
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior *e o menor* valores da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "e o menor é", menor, "\n")  
}  
main()
```

Coleções

Dada uma lista de inteiros, imprima o maior *e o menor* valores da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  menor <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "e o menor é", menor, "\n")  
}  
main()
```

Dada uma lista de inteiros, imprima o maior *e o menor* valores da lista

```
main <- function() {  
  lista <- c(2, 23, -7, 0, 18)  
  maior <- lista[1]  
  menor <- lista[1]  
  i <- 1  
  while (i <= length(lista)) {  
    if (lista[i] > maior) {  
      maior <- lista[i]  
    }  
    if (lista[i] < menor) {  
      menor <- lista[i]  
    }  
    i <- i + 1  
  }  
  cat("O maior número da lista é", maior, "e o menor é", menor, "\n")  
}  
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
```

```
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {

  quantos <- 0

  for (i in 1:n) {
    dados <- sample(1:6, size = 2, replace = TRUE)
    soma <- sum(dados)
    if (soma == 5) {
      quantos <- quantos + 1
    }
  }

  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  i <- 1
  while (i <= n) {

    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  i <- 1
  while (i <= n) {
    dado1 <- sample(1:6, 1)

    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  i <- 1
  while (i <= n) {
    dado1 <- sample(1:6, 1)
    dado2 <- sample(1:6, 1)

    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  i <- 1
  while (i <= n) {
    dado1 <- sample(1:6, 1)
    dado2 <- sample(1:6, 1)
    if (dado1 + dado2 == 5) {

    }
    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))

  i <- 1
  while (i <= n) {
    dado1 <- sample(1:6, 1)
    dado2 <- sample(1:6, 1)
    if (dado1 + dado2 == 5) {
      quantos <- quantos + 1
    }
    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))
  quantos <- 0
  i <- 1
  while (i <= n) {
    dado1 <- sample(1:6, 1)
    dado2 <- sample(1:6, 1)
    if (dado1 + dado2 == 5) {
      quantos <- quantos + 1
    }
    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```

Monte Carlo e dados

Use o método de Monte Carlo para estimar a probabilidade de que, ao jogar dois dados, a soma dos resultados seja 5.

```
library(glue)
main <- function() {
  n <- as.integer(readline("Quantas simulações? "))
  quantos <- 0
  i <- 1
  while (i <= n) {

    result <- sample(1:6, 2, replace=TRUE)
    if (result[1] + result[2] == 5) {
      quantos <- quantos + 1
    }
    i <- i + 1
  }
  cat(glue("A probabilidade de obter 5 é aproximadamente {100 * quantos / n}%"), "\n")
}
main()
```