

MAC 113 — Introdução à Ciência da Computação

Aula 21

Nelson Lago

1º/2025



Previously on MAC113...

Funções vetoriais

- Quando queremos fazer algo envolvendo vetores como um todo, funções vetoriais são muito úteis
- Tão úteis que R possui várias funções vetoriais prontas!

```
x <- c(1, 5, 3)
cat(max(x), mean(x), "\n")
```

5 3

- Tão úteis que, em R, muitas funções e operadores que podem processar dados “simples” são também funções vetoriais!

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
z <- x + y
cat(z, "\n")
```

5 7 9

- Uma função vetorial interessante é **paste()**

- ▶ Ela processa os elementos de um vetor e gera um vetor de strings correspondente

```
frutas <- c("caqui", "maçã", "abacate")  
cat(paste("Eu gosto de", frutas), "\n")
```

Eu gosto de caqui Eu gosto de maçã Eu gosto de abacate

Exercício – cartas do baralho

Usando a função `paste()`, escreva um programa que imprime o nome de todas as cartas do baralho (“ás de paus, 2 de paus,..., 10 de copas, valete de copas...”)

```
main <- function() {  
  cartas <- c()  
  nums <- c("ás", as.character(2:10), "valete", "dama", "rei")  
  naipes <- c("ouros", "copas", "paus", "espadas")  
  for (naipe in naipes) {  
    cartas <- c(cartas, paste(nums, "de", naipe))  
  }  
  cartas <- c(cartas, "coringa vermelho", "coringa preto")  
  cat(cartas, "\n")  
}
```

`main()`

Abstrações

- **As linguagens de programação oferecem **abstrações** básicas**
 - ▶ uma soma é uma série de operações que ocorrem no nível elétrico, mas normalmente pensamos apenas em “+”
 - ▶ variáveis, funções, coleções...
- **Com base nelas, criamos novas abstrações, que se tornam novos conceitos**
 - ▶ Como as peças de um Lego podem ser usadas para construir formas diversas
- **Para objetos computacionais, como por exemplo vectors**
 - ▶ Independentes do problema
 - ▶ Independentes do conteúdo
- **Para objetos relacionados ao problema a ser resolvido**
 - ▶ Variáveis (“dia”, “mês” etc.)
 - ▶ Funções (“`campeonato()`”, “`bissexto()`” etc.)

Abstrações

- **Que tal ao contrário?**
- Além de vectors, R possui **listas**
 - Também é um conjunto ordenado, mas os elementos de uma lista podem ser de tipos diferentes 🙌
- **Um aluno é representado por uma lista (que é um conjunto!)**
 - O primeiro elemento dessa lista é o nome, o segundo é o ID, o terceiro é o endereço e o quarto é o curso

```
mano <- list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos")  
cat(glue("Nome: {mano[[1]]}; ID: {mano[[2]]}; "))  
cat(glue("endereço: {mano[[3]]}; curso: {mano[[4]]}"), "\n")
```

- Acabamos de criar um **novo conceito** (a ideia de “pessoa”) com base em uma **nova abstração**
 - E não vamos pensar muito no fato de essa abstração ser simplesmente uma lista
- Mas e agora, como representar uma lista de pessoas??

- Os elementos de uma lista podem ser de tipos diferentes

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  
)
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos")  
)
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
  
for (i in 1:length(alunos)) {  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
  
for (i in 1:length(alunos)) {  
  aluno <- alunos[[i]]  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
  
for (i in 1:length(alunos)) {  
  aluno <- alunos[[i]]  
  cat(glue("Nome: {aluno[[1]]}; ID: {aluno[[2]]}; "))  
  cat(glue("endereço: {aluno[[3]]}; curso: {aluno[[4]]}"), "\n")  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
  
for (aluno in alunos) {  
  
  cat(glue("Nome: {aluno[[1]]}; ID: {aluno[[2]]}; "))  
  cat(glue("endereço: {aluno[[3]]}; curso: {aluno[[4]]}"), "\n")  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
  
for (aluno in alunos) {  
  
  cat(  
    )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(  
  
  )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(  
  
    )  
  
  )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(itens  
            )  
      )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(itens, aluno          )          )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(itens, aluno, sep=": ")  
  
  )  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(itens, aluno, sep=": "), sep="; ")  
  
}
```

- Os elementos de uma lista podem ser de tipos diferentes
- Cada elemento de uma lista pode, inclusive, ser uma lista!

```
alunos <- list(  
  list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos"),  
  list("Ada Lovelace", 4321, "221B Baker Street", "música computacional")  
)  
itens <- list("Nome", "ID", "endereço", "curso")  
for (aluno in alunos) {  
  
  cat(paste(itens, aluno, sep=": "), sep="; ")  
  cat("\n")  
}
```

AAAAAHHHH!!!!!!

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?!

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**
- Aqui estamos usando a lista para representar uma pessoa, então não faz sentido percorrer a lista

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**
- **Aqui estamos usando a lista para representar uma pessoa, então não faz sentido percorrer a lista**
 - ▶ Não é uma “lista de itens”, mas sim as diferentes informações associadas a uma pessoa (é uma **abstração**: podemos “esquecer” que é uma lista)

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**
- Aqui estamos usando a lista para representar uma pessoa, então não faz sentido percorrer a lista
 - ▶ Não é uma “lista de itens”, mas sim as diferentes informações associadas a uma pessoa (é uma **abstração**: podemos “esquecer” que é uma lista)
 - » (você **pode** percorrer a lista, mas em geral não é isso que você quer)

AAAAAHHHH!!!!!!

- Uma lista em que cada elemento é uma outra lista?!
 - ▶ Sim 😊
- Em geral, fazemos vectors para percorrer todos os elementos
 - ▶ Lista de preços, lista dos vértices de um polígono, lista de operações bancárias...
- **MAS**
- Aqui estamos usando a lista para representar uma pessoa, então não faz sentido percorrer a lista
 - ▶ Não é uma “lista de itens”, mas sim as diferentes informações associadas a uma pessoa (é uma **abstração**: podemos “esquecer” que é uma lista)
 - » (você **pode** percorrer a lista, mas em geral não é isso que você quer)
- Mas faz sentido percorrer a lista de pessoas

- **Um aluno é representado por uma lista (que é um conjunto!)**
 - ▶ O primeiro elemento dessa lista é o nome, o segundo é o ID, o terceiro é o endereço e o quarto é o curso

- Um aluno é representado por uma lista (que é um conjunto!)
 - ▶ O primeiro elemento dessa lista é o nome, o segundo é o ID, o terceiro é o endereço e o quarto é o curso
- Isso é uma **convenção**

- Um aluno é representado por uma lista (que é um conjunto!)
 - ▶ O primeiro elemento dessa lista é o nome, o segundo é o ID, o terceiro é o endereço e o quarto é o curso
- Isso é uma **convenção**
- **MAS**

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

```
mano <- list("Alan Turing", 1234, "Rua dos bobos, 0", "análise de algoritmos")  
  
cat(glue("Nome: {mano[[1]]}; ID: {mano[[2]]}; "))  
cat(glue("endereço: {mano[[3]]}; curso: {mano[[4]]}"), "\n")
```

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

```
mano <- list(nome="Alan Turing", ID=1234,  
             end="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(glue("Nome: {mano[['nome']]]; ID: {mano[['ID']]]; "))  
cat(glue("endereço: {mano[['end']]]; curso: {mano[['curso']]}", "\n"))
```

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

```
mano <- list(nome="Alan Turing", ID=1234,  
             end="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(glue("Nome: {mano[['nome']]}; ID: {mano[['ID']]}; "))  
cat(glue("endereço: {mano[['end']]}; curso: {mano[['curso']]}"), "\n")
```

```
mano <- list(nome="Alan Turing", ID=1234,  
             end="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(glue("Nome: {mano$nome}; ID: {mano$ID}; "))  
cat(glue("endereço: {mano$end}; curso: {mano$curso}"), "\n")
```

Abstrações

- É meio besta usar números para representar conceitos como “nome”, “endereço” etc.
- Listas permitem dar **nomes** a cada um dos seus índices (“posições”)

```
mano <- list(nome="Alan Turing", ID=1234,  
             end="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(glue("Nome: {mano[['nome']]]; ID: {mano[['ID']]]; "))  
cat(glue("endereço: {mano[['end']]]; curso: {mano[['curso']]]}"), "\n")
```

```
mano <- list(nome="Alan Turing", ID=1234,  
             end="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(glue("Nome: {mano$nome}; ID: {mano$ID}; "))  
cat(glue("endereço: {mano$end}; curso: {mano$curso}"), "\n")
```

```
mano <- list(Nome="Alan Turing", ID=1234,  
             endereço="Rua dos bobos, 0", curso="análise de algoritmos")  
cat(paste(names(mano), unlist(mano), sep=": "), sep="; ")  
cat("\n")
```

Cuidado!

Cuidado!

- Os nomes são **strings** (character)

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!
- **MAS**, para facilitar, R permite omitir as aspas

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!
- **MAS**, para facilitar, R permite omitir as aspas
 - ❶ Na definição dos elementos, com “=”

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!
- **MAS**, para facilitar, R permite omitir as aspas
 - ❶ Na definição dos elementos, com “=”
`lista <- list(primeiro=1, segundo=2)`

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!
- **MAS**, para facilitar, R permite omitir as aspas
 - 1 Na definição dos elementos, com “=”

```
lista <- list(primeiro=1, segundo=2)
```
 - 2 Ao acessar um elemento usando a notação com “\$”

Cuidado!

- Os nomes são **strings** (character)
- Ou seja, normalmente ficam entre aspas!
- **MAS**, para facilitar, R permite omitir as aspas

❶ Na definição dos elementos, com “=”

```
lista <- list(primeiro=1, segundo=2)
```

❷ Ao acessar um elemento usando a notação com “\$”

```
nome <- aluno$nome
```

Lembre-se:

Lembre-se:

- Nomes em listas são **opcionais**

Lembre-se:

- Nomes em listas são **opcionais**
- Eles fazem muito sentido quando usamos uma lista para criar uma abstração

Lembre-se:

- Nomes em listas são **opcionais**
- Eles fazem muito sentido quando usamos uma lista para criar uma abstração
- Eles não são muito úteis quando usamos uma lista como uma lista de itens

Lembre-se:

- Nomes em listas são **opcionais**
- Eles fazem muito sentido quando usamos uma lista para criar uma abstração
- Eles não são muito úteis quando usamos uma lista como uma lista de itens
 - ▶ Nos exemplos anteriores, usamos nomes com a abstração pessoa mas não com a lista de pessoas

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”
- Nas listas de R, os índices numéricos continuam valendo

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”
- Nas listas de R, os índices numéricos continuam valendo
- Em dicionários de outras linguagens, geralmente **não!**

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”
- Nas listas de R, os índices numéricos continuam valendo
- Em dicionários de outras linguagens, geralmente **não!**
 - ▶ Nessas linguagens, não há ordem definida em dicionários

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”
- Nas listas de R, os índices numéricos continuam valendo
- Em dicionários de outras linguagens, geralmente **não!**
 - ▶ Nessas linguagens, não há ordem definida em dicionários
 - ▶ Ao percorrer todos os itens do dicionário, eles podem ser processados **em qualquer ordem**

- Em outras linguagens, as coleções mais ou menos similares às listas de R são chamadas **dicionários**
 - ▶ Dada uma “palavra”, o dicionário fornece a “definição”
- Nas listas de R, os índices numéricos continuam valendo
- Em dicionários de outras linguagens, geralmente **não!**
 - ▶ Nessas linguagens, não há ordem definida em dicionários
 - ▶ Ao percorrer todos os itens do dicionário, eles podem ser processados **em qualquer ordem**
 - » *(em versões recentes de python, a ordem é definida: os elementos são processados na ordem em que foram inseridos)*

- Essas são coisas novas?

Abstrações

- Essas são coisas novas?
- **Não!**

- Essas são coisas novas?
- **Não!**
 - 1 Repetições (while, for, operações vetoriais)

- Essas são coisas novas?
- **Não!**
 - ① Repetições (while, for, operações vetoriais)
 - ② Coleções (vectors, lists)

- Essas são coisas novas?
- **Não!**
 - 1 Repetições (while, for, operações vetoriais)
 - 2 Coleções (vectors, lists)
 - 3 Acesso a itens da coleção individualmente (pelo índice numérico ou pelo “nome”)

- Essas são coisas novas?
- **Não!**
 - ① Repetições (while, for, operações vetoriais)
 - ② Coleções (vectors, lists)
 - ③ Acesso a itens da coleção individualmente (pelo índice numérico ou pelo “nome”)
- **Sim!**

- Essas são coisas novas?
- **Não!**
 - ① Repetições (while, for, operações vetoriais)
 - ② Coleções (vectors, lists)
 - ③ Acesso a itens da coleção individualmente (pelo índice numérico ou pelo “nome”)
- **Sim!**
 - ▶ Abstrações mais poderosas

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)

}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {

  }
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(                                     , "\n")
  }
}
```

```
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("                "), "\n")
  }
}
```

```
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("                {notas[[i]]}"), "\n")
  }
}
```

```
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("          : {notas[[i]]}"), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("{          }: {notas[[i]]}"), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("{names(notas)[i]}: {notas[[i]]}"), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (i in 1:length(notas)) {
    cat(glue("{names(notas)[i]}: {notas[[i]]}"), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (nome in names(notas)) {

  }
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (nome in names(notas)) {
    cat(glue("          "), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (nome in names(notas)) {
    cat(glue("{nome}:          "), "\n")
  }
}
```

```
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  for (nome in names(notas)) {
    cat(glue("{nome}: {notas[[nome]]}"), "\n")
  }
}
```

```
main()
```

```
Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
```

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)

}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  cat(                                     , sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  cat(paste(                                     ), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  cat(paste(
    , sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  cat(paste(names(notas)
            , sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

Imprima a lista de notas dos alunos abaixo

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)

  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
           <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

E para acrescentar um aluno?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas[["Rê"]] <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
  cat("Média da turma:"
      , "\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
  cat("Média da turma:", mean(      notas ), "\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
  cat("Média da turma:", mean(unlist(notas)), "\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6

Exercício – Lista de notas

Qual a média da turma?

```
library(glue)
main <- function() {
  notas <- list(Zé=7, Fê=8.5, Má=8.3, Lú=9.1)
  notas$Rê <- 7.6
  cat(paste(names(notas), unlist(notas), sep=": "), sep="\n")
  cat("Média da turma:", mean(unlist(notas)), "\n")
}
main()
```

Zé: 7
Fê: 8.5
Má: 8.3
Lú: 9.1
Rê: 7.6
Média da turma: 8.1

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  logins <- c("turing", "llace", "hopper")  
  senhas <- c("tmachine", "anengine", "business")  
  nomes <- c("Alan Turing", "Ada Lovelace", "Grace Hopper")  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, logins, senhas, nomes)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, logins, senhas, nomes) {  
  i <- 1  
  while (i <= length(senhas)) {  
    if (uid == logins[i] && senha == senhas[i]) {  
      return(nomes[i])  
    }  
    i <- i + 1  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(  
    #cat(saudação(nome), "\n")  
    cat("Bem-vindo,", nome, "\n")  
  )  
}  
checa_login <- function(  
  
  ) {  
  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(  
      uid <- readline("username: ")  
      senha <- readline("senha: ")  
      nome <- checa_login(  
        #cat(saudação(nome), "\n")  
        cat("Bem-vindo,", nome, "\n")  
      )  
    })  
  checa_login <- function(  
    return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing")  
  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(  
    #cat(saudação(nome), "\n")  
    cat("Bem-vindo,", nome, "\n")  
  )  
}  
checa_login <- function(  
  
  ) {  
  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(  
    #cat(saudação(nome), "\n")  
    cat("Bem-vindo,", nome, "\n")  
  )  
}  
checa_login <- function(  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(  
    #cat(saudação(nome), "\n")  
    cat("Bem-vindo,", nome, "\n")  
  )  
}  
checa_login <- function(  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(                                     ) {  
  
  
  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (  
        
    ) {  
        
    }  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (uid ==  
    ) {  
      }  
    }  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (uid == user$login  
        ) {  
      }  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (uid == user$login &&  
        ) {  
      }  
    }  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (uid == user$login && senha == user$senha) {  
  
    }  
  }  
  return("")  
}
```

Exercício – Sistema de *login*

Modifique o código abaixo para usar uma lista como abstração para usuário

```
main <- function() {  
  usuários <- list(  
    list(login="turing", senha="tmachine", nome="Alan Turing"),  
    list(login="llace", senha="anengine", nome="Ada Lovelace"),  
    list(login="hopper", senha="business", nome="Grace Hopper")  
  )  
  uid <- readline("username: ")  
  senha <- readline("senha: ")  
  nome <- checa_login(uid, senha, usuários)  
  #cat(saudação(nome), "\n")  
  cat("Bem-vindo,", nome, "\n")  
}  
checa_login <- function(uid, senha, usuários) {  
  for (user in usuários) {  
    if (uid == user$login && senha == user$senha) {  
      return (user$nome)  
    }  
  }  
  return("")  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
if (opção == 1) {  
  
  
} else if (opção == 2) {  
  
  
} else if (opção == 3) {  
  
  
} else if (opção == 4) {  
  
  
} else {  
  
  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
while (          ) {  
    if (opção == 1) {  
  
    } else if (opção == 2) {  
  
    } else if (opção == 3) {  
    } else if (opção == 4) {  
    } else {  
  
    }  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
while (opção != 4) {  
    if (opção == 1) {  
  
    } else if (opção == 2) {  
  
    } else if (opção == 3) {  
    } else if (opção == 4) {  
    } else {  
  
    }  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
while (opção != 4) {  
  opção <- as.integer(readline("Escolha a opção desejada: "))  
  if (opção == 1) {  
  
  
  } else if (opção == 2) {  
  
  
  } else if (opção == 3) {  
  
  
  } else if (opção == 4) {  
  
  
  } else {  
  
  
  }  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <-  
while (opção != 4) {  
  opção <- as.integer(readline("Escolha a opção desejada: "))  
  if (opção == 1) {  
  
  } else if (opção == 2) {  
  
  } else if (opção == 3) {  
  
  } else if (opção == 4) {  
  
  } else {  
  
  }  
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {

  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {

  } else {

  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {

  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {

  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {

  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")

  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    <- fone
  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    <- fone
  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos <- fone
  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {

  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
  } else if (opção == 3) {
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", " ", "\n")
  } else if (opção == 3) {
    # Listar todos os telefones
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {

  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {
    cat(paste(
    )
    )
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {
    cat(paste(names(contatos), "\n"))
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {
    cat(paste(names(contatos), unlist(contatos)          )          )
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {
    cat(paste(names(contatos), unlist(contatos), sep=": ")
  )
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```

Exercício – agenda telefônica

Faça um programa de agenda de telefones. O programa deve perguntar ao usuário a ação desejada (1: cadastrar um novo telefone; 2: procurar um telefone pelo nome; 3: listar todos os telefones; 4: sair) e agir de acordo.

```
contatos <- list()
opção <- 0 # Qualquer valor diferente de 4
while (opção != 4) {
  opção <- as.integer(readline("Escolha a opção desejada: "))
  if (opção == 1) {
    nome <- readline("Nome? ")
    fone <- readline("Telefone? ")
    contatos[[nome]] <- fone
  } else if (opção == 2) {
    nome <- readline("Nome? ")
    cat("0 telefone de", nome, "é", contatos[[nome]], "\n")
  } else if (opção == 3) {
    cat(paste(names(contatos), unlist(contatos), sep=": "), sep="\n")
  } else if (opção == 4) {
    cat("Obrigado por usar a agenda telefônica!\n")
  } else {
    cat("Oops! Opção inválida, tente novamente\n")
  }
}
```