MAC 113 — Introdução à Ciência da Computação

Aula 1

Nelson Lago

1°/2025

Meta

- **Informações sobre a disciplina no eDisciplinas**
  - Datas de provas e critérios de avaliação
  - Slides das aulas (a cada aula)
  - Exercícios para praticar
  - Entrega dos exercícios que valem nota
  - Links para muitos materiais adicionais interessantes
  - Fórum de discussões e informações sobre a monitoria
  - O eDisciplinas pode enviar avisos por email; não deixe de checar sua caixa postal regularmente

Fazer muitos exercícios é fundamental

MAC113 – Intro à Computação para C. Humanas

(adm noturno 2025)

edisciplinas.usp.br/course/view.php?id=127660

Plágio/cola/etc. não dá, né?

Plágio/cola/etc. não dá, né?

Leia o item sobre “Plágio++” no eDisciplinas

Oncotô?

Computação:

Resolução de problemas do mundo real com os recursos que o computador oferece

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)
  - Com exemplos usando ciência de dados

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)
  - Com exemplos usando ciência de dados
- **Mas não é um "curso de R"!**

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)
  - Com exemplos usando ciência de dados
- **Mas não é um "curso de R"!**
  - O ensino de uma linguagem de programação é uma ferramenta didática e um bônus concreto de aprendizagem

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)
  - Com exemplos usando ciência de dados
- **Mas não é um "curso de R"!**
  - O ensino de uma linguagem de programação é uma ferramenta didática e um bônus concreto de aprendizagem
- **Não é um curso de ciência de dados!**

Objetivos

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
  - Aprender a escrever programas de computador (na linguagem **R**)
  - Com exemplos usando ciência de dados
- **Mas não é um "curso de R"!**
  - O ensino de uma linguagem de programação é uma ferramenta didática e um bônus concreto de aprendizagem
- **Não é um curso de ciência de dados!**
  - A ciência de dados também é uma ferramenta didática e um bônus concreto da aprendizagem

Resolução de problemas do mundo real com os recursos que o computador oferece

Programando

Programar envolve

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*
     - » *E não tem vontade própria! ("o sistema não permite...")*

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*
     - » *E não tem vontade própria! ("o sistema não permite...")*
3. ***Implementar* o algoritmo em uma linguagem de programação**
   - Gerando um *programa* que pode ser *executado* para solucionar o problema

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*
     - » *E não tem vontade própria! ("o sistema não permite...")*
3. ***Implementar* o algoritmo em uma linguagem de programação**
   - Gerando um *programa* que pode ser *executado* para solucionar o problema
     - » *Uma receita de bolo em alemão é útil? Para quem?*

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - » *Cálculos?*
   - » *Armazenamento e recuperação de dados?*
   - » *Processamento de multimídia?*
   - » *...*
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - Sequência finita de passos bem definidos baseados em um conjunto limitado (vocabulário) de operações possíveis
     - » *O algoritmo é abstrato (como a planta de um prédio ou uma receita de bolo)*
     - » *E não tem vontade própria! ("o sistema não permite...")*
3. ***Implementar* o algoritmo em uma linguagem de programação**
   - Gerando um *programa* que pode ser *executado* para solucionar o problema
     - » *Uma receita de bolo em alemão é útil? Para quem?*
4. **Testar o programa**

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 5 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} {\scriptstyle 1} \\ 123 \\ \times\, 45 \\ \hline 5 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 15 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} {\scriptstyle 1} \\ 123 \\ \times\, 45 \\ \hline 15 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ + \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 2+ \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} {\scriptstyle 1} \\ 123 \\ \times\, 45 \\ \hline 615 \\ 2+ \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 92+ \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 492+ \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 492+ \\ \hline \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$
\begin{array}{r}
123 \\
\times\, 45 \\
\hline
615 \\
492+ \\
\hline
5
\end{array}
$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 492+ \\ \hline 35 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline 615 \\ 492+ \\ \hline 535 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$\begin{array}{r} 123 \\ \times\, 45 \\ \hline {\scriptstyle 1}615 \\ 492+ \\ \hline 535 \end{array}$$

Um exemplo de algoritmo

Se eu sou capaz se multiplicar e somar dois dígitos,
posso multiplicar quaisquer inteiros:

$$
\begin{array}{r}
123 \\
\times\, 45 \\
\hline
615 \\
492+ \\
\hline
5535
\end{array}
$$

Programando

Programar também envolve

Programando

Programar também envolve

1. **Construir software capaz de lidar com dificuldades especiais**
   - Desempenho (sistemas de tempo real, HPC...)
   - Grandes quantidades de dados (big data, aprendizado de máquina...)
   - Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
   - Segurança (ataques cibernéticos, privacidade...)

Programando

Programar também envolve

1. **Construir software capaz de lidar com dificuldades especiais**
   - Desempenho (sistemas de tempo real, HPC...)
   - Grandes quantidades de dados (big data, aprendizado de máquina...)
   - Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
   - Segurança (ataques cibernéticos, privacidade...)
2. **Gerenciar software de grande porte, composto por várias partes que precisam ser integradas**

Programando

Programar também envolve

1. **Construir software capaz de lidar com dificuldades especiais**
   - Desempenho (sistemas de tempo real, HPC...)
   - Grandes quantidades de dados (big data, aprendizado de máquina...)
   - Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
   - Segurança (ataques cibernéticos, privacidade...)
2. **Gerenciar software de grande porte, composto por várias partes que precisam ser integradas**
3. **Gerenciar o processo e a equipe de desenvolvimento**

Programando

Programar também envolve

1. **Construir software capaz de lidar com dificuldades especiais**
   - Desempenho (sistemas de tempo real, HPC...)
   - Grandes quantidades de dados (big data, aprendizado de máquina...)
   - Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
   - Segurança (ataques cibernéticos, privacidade...)
2. **Gerenciar software de grande porte, composto por várias partes que precisam ser integradas**
3. **Gerenciar o processo e a equipe de desenvolvimento**
4. **Comunicar-se com clientes e usuários**

Programando

Programar também envolve

1. **Construir software capaz de lidar com dificuldades especiais**
   - Desempenho (sistemas de tempo real, HPC...)
   - Grandes quantidades de dados (big data, aprendizado de máquina...)
   - Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
   - Segurança (ataques cibernéticos, privacidade...)
2. **Gerenciar software de grande porte, composto por várias partes que precisam ser integradas**
3. **Gerenciar o processo e a equipe de desenvolvimento**
4. **Comunicar-se com clientes e usuários**

Programando

Para ser útil, um programa geralmente

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado
3. **"Faz alguma coisa" com esse resultado**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado
3. **"Faz alguma coisa" com esse resultado**
   - Mostra para o usuário

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado
3. **"Faz alguma coisa" com esse resultado**
   - Mostra para o usuário
   - **Utiliza como dado para fazer outra coisa**

Lembre-se:

Lembre-se:

O computador só faz o que você manda!

Lembre-se:

O computador só faz o que você manda!

(não o que você quer)

Lembre-se:

O computador só faz o que você manda!

(não o que você quer)

Se você esquecer um passo, ele obedece!

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema
  - Conhecimento prévio

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema
  - Conhecimento prévio
  - Gosto pessoal

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema
  - Conhecimento prévio
  - Gosto pessoal
  - ...

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema
  - Conhecimento prévio
  - Gosto pessoal
  - ...
- **R (r-project.org) é uma linguagem amplamente usada em estatística e economia**

Linguagens de programação

- **Um programa de computador é uma descrição detalhada dos algoritmos usados para resolver um problema**
  - Essa descrição é feita usando uma linguagem de programação
    - » *Criada para facilitar a "comunicação" entre o programador e o computador*
    - » *Ou melhor, para facilitar a expressão de soluções algorítmicas usando o computador*
- **Existem centenas de linguagens de programação**
  - Vantagens e desvantagens ↔ tipo de problema
  - Conhecimento prévio
  - Gosto pessoal
  - ...
- **R (r-project.org) é uma linguagem amplamente usada em estatística e economia**
  - Coincidentemente, é a linguagem que vamos usar neste curso 😉

IDEs

- Em geral, um programa é um **arquivo de texto**

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**
  - Visual Studio Code + plugin REditorSupport
  - StatET for R (baseado no Eclipse)
  - ...

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**
  - Visual Studio Code + plugin REditorSupport
  - StatET for R (baseado no Eclipse)
  - ...
  - O mais popular é o R Studio

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**
  - Visual Studio Code + plugin REditorSupport
  - StatET for R (baseado no Eclipse)
  - ...
  - O mais popular é o R Studio
- **Durante as aulas, usaremos o google colab**
  - colab.research.google.com

IDEs

- **Em geral, um programa é um arquivo de texto**
  - Arquivos de texto podem ser escritos em português, chinês, alemão...
  - Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**
  - Visual Studio Code + plugin REditorSupport
  - StatET for R (baseado no Eclipse)
  - ...
  - O mais popular é o R Studio
- **Durante as aulas, usaremos o google colab**
  - colab.research.google.com
- **Mas você pode usar qualquer outro**

Configurando o ambiente

https://colab.research.google.com

Introdução · Most Visited · Smart Bookmarks · Getting Started · Slashdot · Getting Started · Most Visited · Outros favoritos

Olá, este é o Colaboratory

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Compartilhar

Índice

Vamos começar
Ciência de dados
Machine learning
Mais recursos
Exemplos em destaque
+ Seção

Comandos + Código + Texto Copiar

Conectar

Conheça o Colab

(Novidade) Teste a API Gemini

- Generate a Gemini API key
- Talk to Gemini with the Speech-to-Text API
- Gemini API: Quickstart with Python
- Gemini API code sample
- Compare Gemini with ChatGPT
- More notebooks

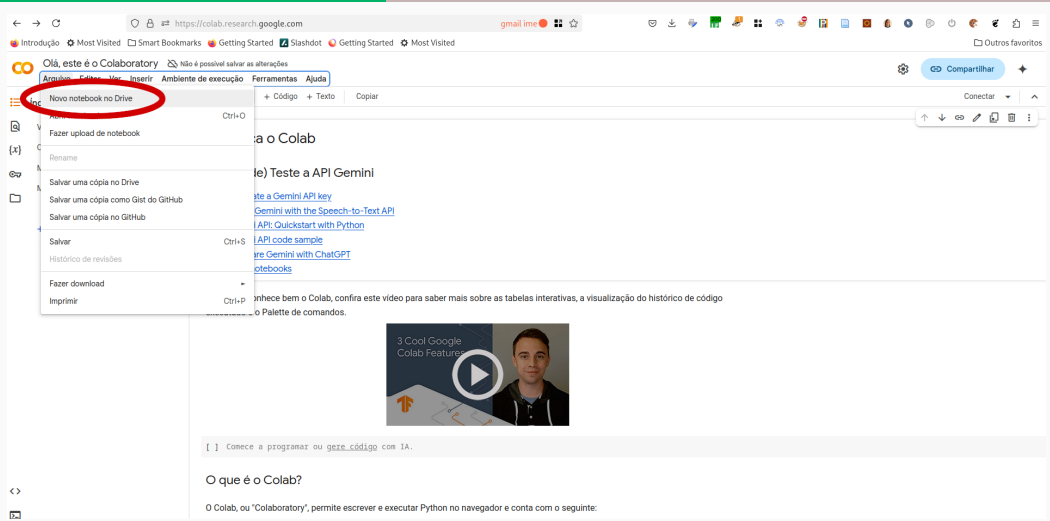
Se você já conhece bem o Colab, confira este vídeo para saber mais sobre as tabelas interativas, a visualização do histórico de código executado e o Palette de comandos.

[ ] Comece a programar ou gere código com IA.

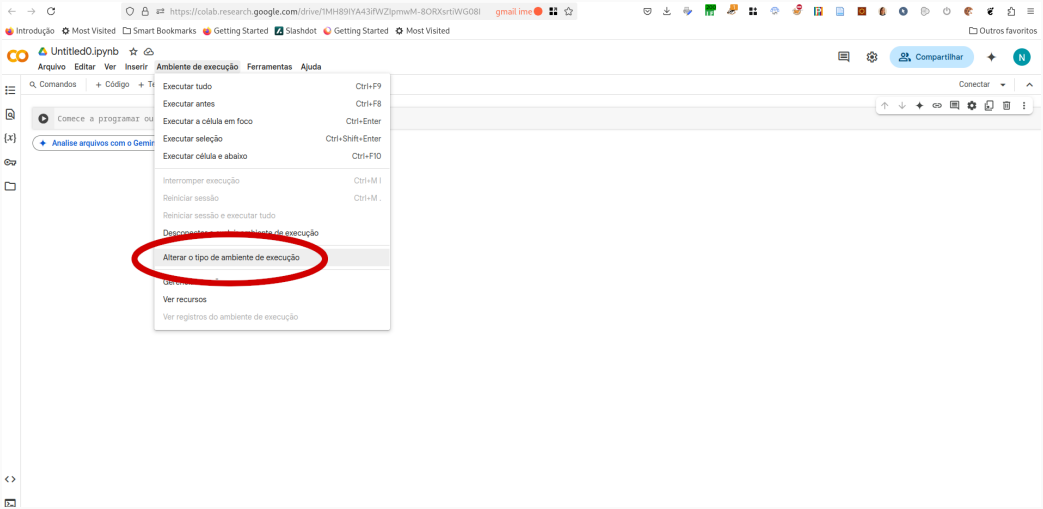
O que é o Colab?

O Colab, ou "Colaboratory", permite escrever e executar Python no navegador e conta com o seguinte:

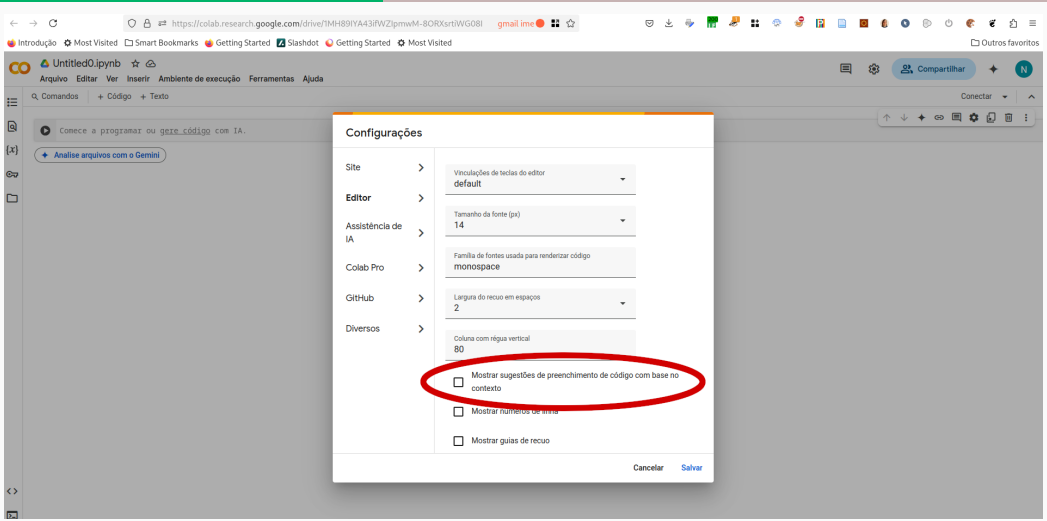
Configurando o ambiente



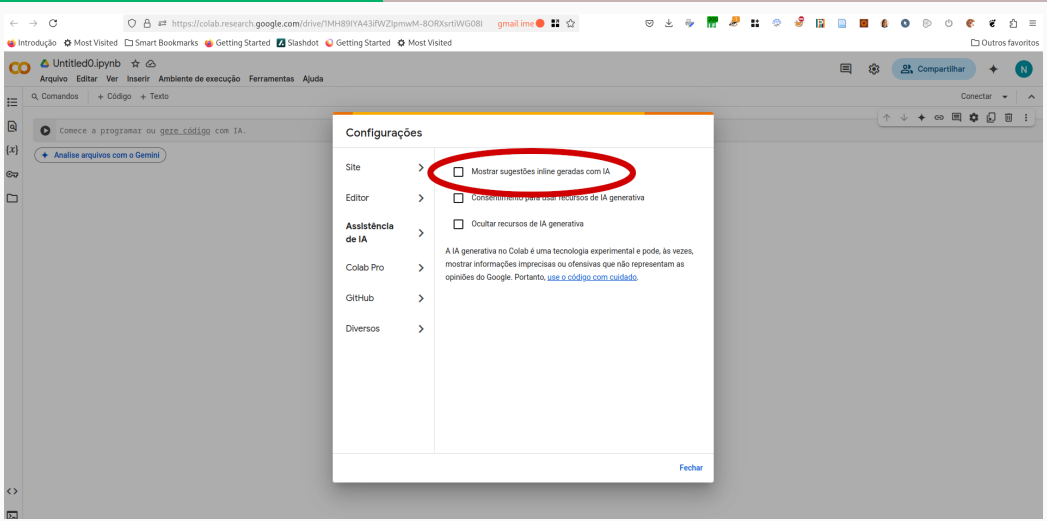
Configurando o ambiente



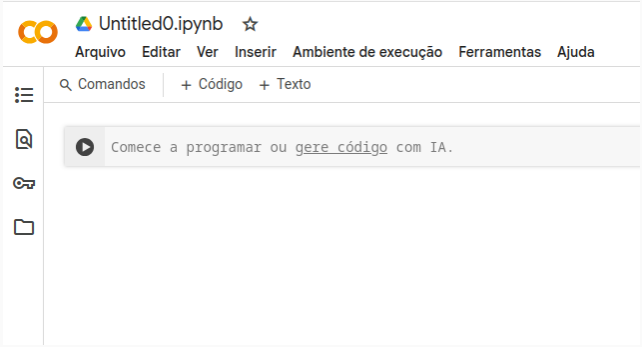
Configurando o ambiente



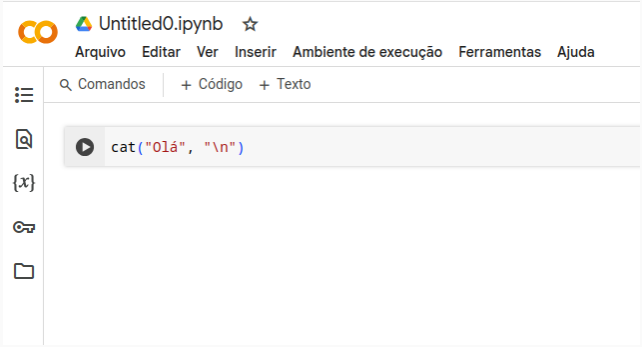
Configurando o ambiente



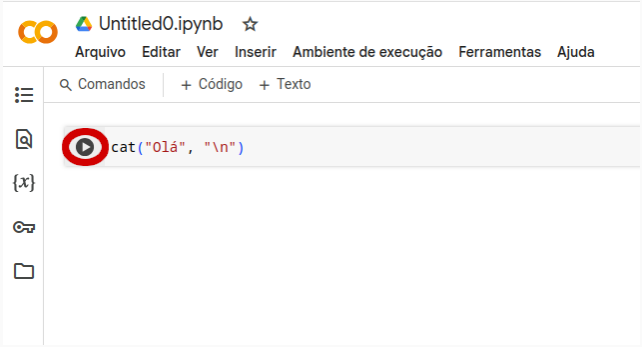
cat()


Untitled0.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos
+ Código
+ Texto
Comece a programar ou gere código com IA.

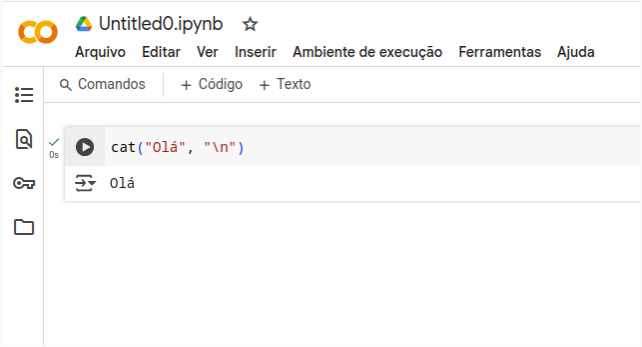
cat()


Untitled0.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos + Código + Texto
cat("Olá", "\n")

cat()


Untitled0.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos
+ Código
+ Texto
cat("Olá", "\n")

cat()


Untitled0.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Comandos + Código + Texto
0s
cat("Olá", "\n")
Olá

cat()

```
cat("Olá!", "\n")
```

cat()

```
cat("Olá!", "\n")
```

```
Olá
```

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: $2 + 3$

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: $2 + 3$
  - Exemplo: "Oi galera!"

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: 2 + 3 + 7

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: $2 + 3$
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7$
  - Exemplo: $\frac{2+3+7}{6}$

Expressões em R

- **A maioria das coisas em R são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: $2 + 3$
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7$
  - Exemplo: $\frac{2+3+7}{6}$

Expressões aritméticas

$$2 + 3 + 7$$

Expressões aritméticas

$$2 + 3 + 7$$

```
cat(2 + 3 + 7, "\n")
```

Expressões aritméticas

$$2 + 3 + 7$$

```
cat(2 + 3 + 7, "\n")
```

```
12
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
cat(2 + 3 + 7 / 6, "\n")
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
cat(2 + 3 + 7 / 6, "\n")
```

```
6.166667
```

Oops!

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
cat((2 + 3 + 7) / 6, "\n")
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
cat((2 + 3 + 7) / 6, "\n")
```

```
2
```

Aêh!

Precedência de operadores

operador	descrição	associatividade
`()`	parênteses	da esquerda para a direita
`^`	exponenciação	da direita para a esquerda
`+ -`	positivo e negativo unário	da esquerda para a direita
`%% %/%`	resto e divisão inteira	da esquerda para a direita
`* /`	multiplicação e divisão	da esquerda para a direita
`+ -`	soma e subtração	da esquerda para a direita

Brincando com dígitos

- O operador `%/%` faz uma divisão inteira

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
  - `12345 %% 10 = 5`

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
  - `12345 %% 10 = 5`
    - » *Qualquer número ... xyz pode ser escrito como ... $xy * 10 + z$*

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
  - `12345 %% 10 = 5`
    - » *Qualquer número ... xyz pode ser escrito como ... $xy * 10 + z$*
    - » *... $xy * 10$ é múltiplo de 10, então o resto da divisão de ... xyz por 10 é z*

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
  - `12345 %% 10 = 5`
    - » *Qualquer número ... xyz pode ser escrito como ... $xy * 10 + z$*
    - » *... $xy * 10$ é múltiplo de 10, então o resto da divisão de ... xyz por 10 é z*
    - » *(e, como visto acima, z sempre é menor que 10)*

Brincando com dígitos

- **O operador `%/%` faz uma divisão inteira**
  - `13 %/% 5 = 2`
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
  - `12345 %/% 10 = 1234`
- **O operador `%%` fornece o resto da divisão**
  - `13 %% 5 = 3`
    - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
  - `12345 %% 10 = 5`
    - » *Qualquer número ... xyz pode ser escrito como ... $xy * 10 + z$*
    - » *... $xy * 10$ é múltiplo de 10, então o resto da divisão de ... xyz por 10 é z*
    - » *(e, como visto acima, z sempre é menor que 10)*

(Não é exatamente "divisão inteira" nem "resto da divisão", mas ok)

Brincando com dígitos

Encontre o último dígito de um número:

```
cat(2407 %% 10, "\n")
cat(110 %% 10, "\n")
cat(3 %% 10, "\n")
cat(0 %% 10, "\n")
```

Brincando com dígitos

Encontre o último dígito de um número:

```
cat(2407 %% 10, "\n")
cat(110 %% 10, "\n")
cat(3 %% 10, "\n")
cat(0 %% 10, "\n")
```

```
7
```

Brincando com dígitos

Encontre o último dígito de um número:

```
cat(2407 %% 10, "\n")
cat(110 %% 10, "\n")
cat(3 %% 10, "\n")
cat(0 %% 10, "\n")
```

```
7
0
```

Brincando com dígitos

Encontre o último dígito de um número:

```
cat(2407 %% 10, "\n")
cat(110 %% 10, "\n")
cat(3 %% 10, "\n")
cat(0 %% 10, "\n")
```

```
7
0
3
```

Brincando com dígitos

Encontre o último dígito de um número:

```
cat(2407 %% 10, "\n")
cat(110 %% 10, "\n")
cat(3 %% 10, "\n")
cat(0 %% 10, "\n")
```

```
7
0
3
0
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("a:", (n%/%10) * 10, "\n")
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("a:", (n%/%10) * 10, "\n")
cat("b:", n %% 10, "\n")
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("a:", (n%/%10) * 10, "\n")
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("a:", (n%/%10) * 10, "\n")
cat("b:", n - a, "\n")
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("b:", n %% 10, "\n")
```

Brincando com dígitos

Dado um número n, transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

```
cat("b:", n %% 10, "\n")
cat("a:", n - b, "\n")
```

Tipos

```
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

Tipos

```
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
```

Tipos

```r
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
```

Tipos

```r
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
[Erro]
```

Tipos

```
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
[Erro]
```

Tipos

```r
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

5
2 + 3
[Erro]
0.3333333

Tipos

```r
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
[Erro]
0.3333333
5
```

Tipos

```r
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
[Erro]
0.3333333
5
0.333333333333333
```

Tipos

```
cat(2 + 3, "\n")
cat("2 + 3", "\n")
cat("2" + "3", "\n")
cat(1/3, "\n")
options(digits=15)
cat(2 + 3, "\n")
cat(1/3, "\n")
```

```
5
2 + 3
[Erro]
0.3333333
5
0.333333333333333
```

Tipos

- **Obviamente, `2 + 3` não é a mesma coisa que `"2 + 3"`**
  - Um é uma operação, o outro é um texto

Tipos

- **Obviamente, `2 + 3` não é a mesma coisa que `"2 + 3"`**
  - Um é uma operação, o outro é um texto
    - » `2 + 3` **e** `2.4 + 2.6` *também não são a mesma operação (somar números inteiros e somar números não-inteiros são operações diferentes, mesmo que chamemos ambas de "+")*

Tipos

- **Obviamente, `2 + 3` não é a mesma coisa que `"2 + 3"`**
  - Um é uma operação, o outro é um texto
    - » `2 + 3` *e* `2.4 + 2.6` *também não são a mesma operação (somar números inteiros e somar números não-inteiros são operações diferentes, mesmo que chamemos ambas de "+")*
    - » *Imprimir um número como* $\frac{1}{3}$ *também não é a mesma operação que imprimir um número inteiro (é preciso "decidir" o que fazer com o arredondamento)*

Tipos

- **Obviamente, `2 + 3` não é a mesma coisa que `"2 + 3"`**
  - Um é uma operação, o outro é um texto
    - » *`2 + 3` e `2.4 + 2.6` também não são a mesma operação (somar números inteiros e somar números não-inteiros são operações diferentes, mesmo que chamemos ambas de "+")*
    - » *Imprimir um número como $\frac{1}{3}$ também não é a mesma operação que imprimir um número inteiro (é preciso "decidir" o que fazer com o arredondamento)*
- **Se solicitarmos ao computador que calcule $\frac{2}{3}$, quanto de memória vai ser necessário para armazenar o resultado?**

Tipos

- **Obviamente, `2 + 3` não é a mesma coisa que `"2 + 3"`**
  - Um é uma operação, o outro é um texto
    - » *`2 + 3` e `2.4 + 2.6` também não são a mesma operação (somar números inteiros e somar números não-inteiros são operações diferentes, mesmo que chamemos ambas de "+")*
    - » *Imprimir um número como $\frac{1}{3}$ também não é a mesma operação que imprimir um número inteiro (é preciso "decidir" o que fazer com o arredondamento)*
- **Se solicitarmos ao computador que calcule $\frac{2}{3}$, quanto de memória vai ser necessário para armazenar o resultado?**
- **E $\sqrt{2}$?**

Tipos

- Existem *tipos de dados* diferentes em R

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*
    - » *Números entre ~ -2 bilhões e ~ 2 bilhões*

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*
    - » *Números entre ~-2 bilhões e ~2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*
    - » *Números entre ~ -2 bilhões e ~ 2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*
    - » *Números entre ~−2 bilhões e ~2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - »
```
options(digits=18)
cat(5^0.5 * 5^0.5, "\n")
5.00000000000000089
```

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou -437
    - » *Não são comumente usados em R*
    - » *Números entre ~−2 bilhões e ~2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - » `options(digits=18)`
      `cat(5^0.5 * 5^0.5, "\n")`
      `5.00000000000000089` *Ok, números irracionais...*

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou −437
    - » *Não são comumente usados em R*
    - » *Números entre ∼−2 bilhões e ∼2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - »
```
options(digits=18)
cat(5^0.5 * 5^0.5, "\n")
5.00000000000000089
```
Ok, números irracionais...
```
cat(0.1 + 0.1 + 0.1, "\n")
0.300000000000000044
```

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou −437
    - » *Não são comumente usados em R*
    - » *Números entre ∼−2 bilhões e ∼2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - »
```
options(digits=18)
cat(5^0.5 * 5^0.5, "\n")
5.00000000000000089
```
Ok, números irracionais...
```
cat(0.1 + 0.1 + 0.1, "\n")
0.300000000000000044
```
😱

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou −437
    - » *Não são comumente usados em R*
    - » *Números entre ∼−2 bilhões e ∼2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - » `options(digits=18)`

      `cat(5^0.5 * 5^0.5, "\n")`

      `5.00000000000000089` *Ok, números irracionais...*

      `cat(0.1 + 0.1 + 0.1, "\n")`

      `0.300000000000000044` 😱
  - Textos (“strings” ou “cadeias de caracteres”), como `"Oi galera!"`

Tipos

- **Existem *tipos de dados* diferentes em R**
  - Números inteiros, como 2 ou −437
    - » *Não são comumente usados em R*
    - » *Números entre ~−2 bilhões e ~2 bilhões*
  - Números (potencialmente) não-inteiros (“números de ponto flutuante”), como 2.0 ou 6.166666666666667
    - » *O tipo mais comum em R*
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - »
```
options(digits=18)
cat(5^0.5 * 5^0.5, "\n")
```
```
5.00000000000000089
```
Ok, números irracionais...
```
cat(0.1 + 0.1 + 0.1, "\n")
```
```
0.300000000000000044
```
😱
  - Textos (“strings” ou “cadeias de caracteres”), como `"Oi galera!"`
- **R “sabe” quais operações podem ser realizadas com cada tipo**

Nomes (variáveis)

- Ao programar, preferimos pensar no problema a ser resolvido e não nas idiossincrasias do computador
- Linguagens de programação de alto nível procuram oferecer os recursos para isso
- Uma das coisas mais importantes para esse fim é utilizar *nomes*

Nomes (variáveis)

x ← 5 (atribuição)

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Nomes (variáveis)

x ⟵ 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:

Nomes (variáveis)

x ⟵ 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:

```
x = x + 1
```

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:

```
x = x + 1
```

AAAAAHHHH!!!!!!!

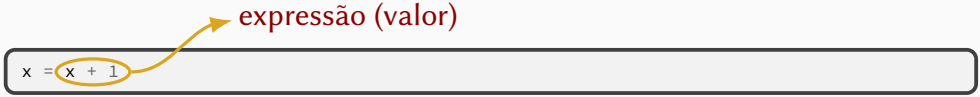
Nomes (variáveis)

x ⟵ 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:



AAAAAHHHH!!!!!!!

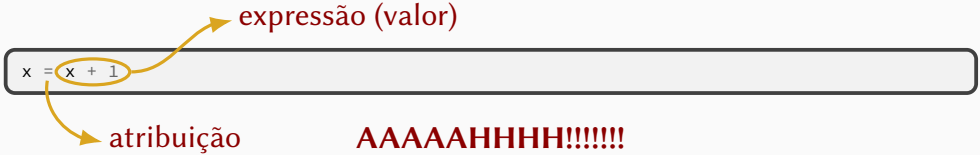
Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:



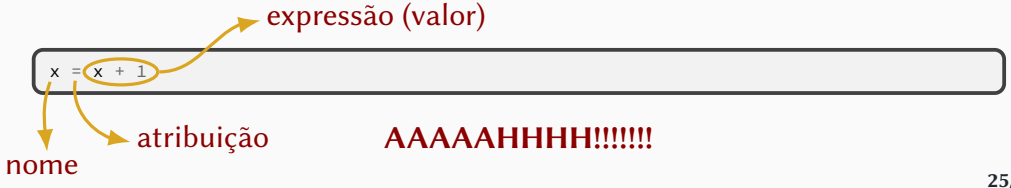
Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em R com "<-":

```
x <- 5
```

Muitas linguagens utilizam = para fazer atribuição, e R aceita também:



Nomes (variáveis)

```r
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

Nomes (variáveis)

```r
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

---

numeric

Nomes (variáveis)

```r
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

---

numeric
character

Nomes (variáveis)

```r
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

---

numeric
character
Error in cat(class(x)) : objeto 'x' não encontrado

Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

---

numeric
character
Error in cat(class(x)) : objeto 'x' não encontrado

Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

```
numeric
character
Error in cat(class(x)) : objeto 'x' não encontrado
numeric
```

Nomes (variáveis)

```r
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

---

numeric
character
Error in cat(class(x)) : objeto 'x' não encontrado
numeric
integer

Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("Olar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "Olar!"
cat(class(x), "\n")
```

```
numeric
character
Error in cat(class(x)) : objeto 'x' não encontrado
numeric
integer
character
```

Como o R faz a distinção entre nomes e strings?

Como o R faz a distinção entre nomes e strings?

Aspas!

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$-x^2 + 3x + 4 = 0$$

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$-x^2 + 3x + 4 = 0$$

```
cat((-3 + sqrt(3^2 - 4 * -1 * 4)) / -2,
    (-3 - sqrt(3^2 - 4 * -1 * 4)) / -2, "\n")
```

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$-x^2 + 3x + 4 = 0$$

```
cat((-3 + sqrt(3^2 - 4 * -1 * 4)) / -2,
    (-3 - sqrt(3^2 - 4 * -1 * 4)) / -2, "\n")
```

```
-1 4
```

Nomes (variáveis)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$-x^2 + 3x + 4 = 0$$

```
cat((-3 + sqrt(3^2 - 4 * -1 * 4)) / -2,
    (-3 - sqrt(3^2 - 4 * -1 * 4)) / -2, "\n")
```

```
-1 4
```

Nomes (variáveis)

```r
a <- -1
b <- 3
c <- 4
delta <- b^2 - 4 * a * c
raiz1 <- (-1 * b + sqrt(delta)) / 2 * a
raiz2 <- (-1 * b - sqrt(delta)) / 2 * a
cat("As raízes são", raiz1, "e", raiz2, "\n")
```

Nomes (variáveis)

```r
a <- -1
b <- 3
c <- 4
delta <- b^2 - 4 * a * c
raiz1 <- (-1 * b + sqrt(delta)) / 2 * a
raiz2 <- (-1 * b - sqrt(delta)) / 2 * a
cat("As raízes são", raiz1, "e", raiz2, "\n")
```

```
As raízes são -1 e 4
```

Nomes (variáveis)

```r
# Calcula raízes da equação de segundo grau
a <- -1
b <- 3
c <- 4
delta <- b^2 - 4 * a * c  # Não pode ser menor que zero!
raiz1 <- (-1 * b + sqrt(delta)) / 2 * a
raiz2 <- (-1 * b - sqrt(delta)) / 2 * a
cat("As raízes são", raiz1, "e", raiz2, "\n")
```

```
As raízes são -1 e 4
```

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação “real” que está em algum lugar na memória do computador)
    - » *Como na matemática!*

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação “real” que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de “variáveis”

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de "variáveis"
  - Acima, definimos o valor dos nomes (variáveis) com comandos de atribuição (`a <- -1`) fixos. Como fazer se queremos valores que não sejam fixos?

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de "variáveis"
  - Acima, definimos o valor dos nomes (variáveis) com comandos de atribuição (`a <- -1`) fixos. Como fazer se queremos valores que não sejam fixos?
  - `readline()`

Primeiro programa – bully

```
idade <- readline(prompt="Informe sua idade: ")
```

Primeiro programa – bully

```r
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

Primeiro programa – bully

```
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

```
Informe sua idade: 23
```

Primeiro programa – bully

```
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

```
Informe sua idade: 23
Você tem só 23 anos?!?!
```

Primeiro programa – bully

```
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

Informe sua idade: 23
Você tem só 23 anos?!?!
Error in 2 * idade : argumento não-numérico para operador binário

Primeiro programa – bully

```r
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

Primeiro programa – bully

```r
idade <- as.integer(readline(prompt="Informe sua idade: "))

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```

Primeiro programa – bully

```r
idade <- readline(prompt="Informe sua idade: ")

cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * as.integer(idade), "anos!", "\n")
```

Primeiro programa – bully

```r
idade <- readline(prompt="Informe sua idade: ")
idade <- as.integer(idade)
cat("Você tem só", idade, "anos?!?!", "\n")
cat("Nossa, você aparenta ter", 2 * idade, "anos!", "\n")
```