

# MAC 113 – Introdução à Ciência da Computação

## Aula 1

---

Nelson Lago

1º/2025



- **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](https://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**

## **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

Programar envolve

## 1 Compreender um problema em termos computacionais

- » *Cálculos?*
- » *Processamento de multimídia?*
- » *Armazenamento e recuperação de dados?*
- » *...*













# Um exemplo de algoritmo

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

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

# Um exemplo de algoritmo

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

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

# Um exemplo de algoritmo

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

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

# Um exemplo de algoritmo

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

$$\begin{array}{r} 123 \\ \times 45 \\ \hline 615 \\ 492+ \\ \hline 5565 \end{array}$$

# Um exemplo de algoritmo

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

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

# Um exemplo de algoritmo

Se eu sou capaz de 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

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...)

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

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**

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**

Para ser útil, um programa geralmente

Para ser útil, um programa geralmente

## 1 **Obtém dados**

Para ser útil, um programa geralmente

- 1 **Obtém dados**
- 2 **“Faz alguma coisa” com esses dados**

Para ser útil, um programa geralmente

- 1 **Obtém dados**
- 2 **“Faz alguma coisa” com esses dados**
  - ▶ Gerando um resultado

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**

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

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 😊

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

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

- 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

- 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

- 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!**

- 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

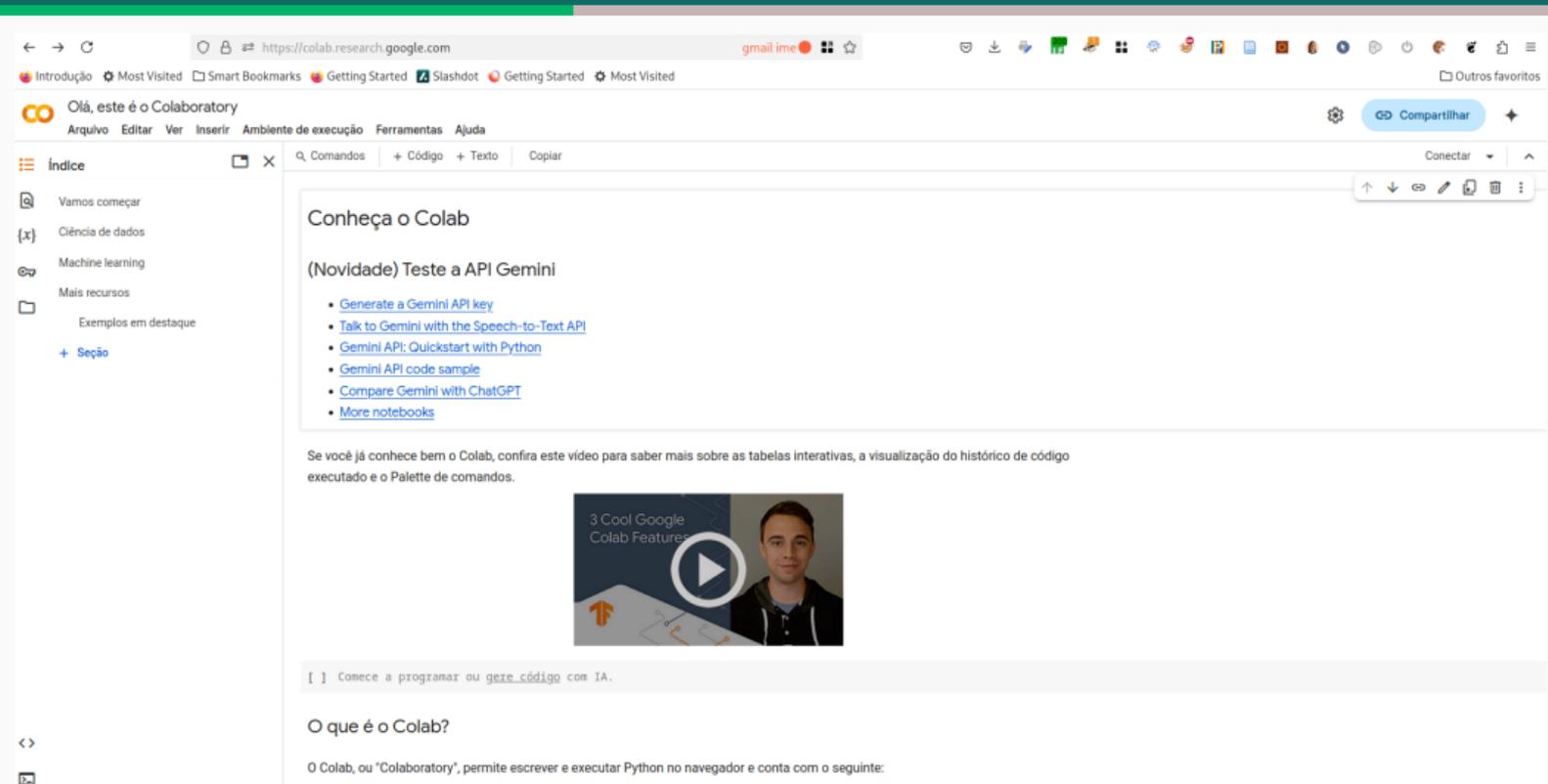
- **Em geral, um programa é um **arquivo de texto****
  - ▶ Arquivos de texto podem ser escritos em português, chinês, alemão...
  - ▶ Ou em uma linguagem de programação
- **É possível usar qualquer editor de texto**
- **MAS!**
- **É muito mais confortável usar editores especializados**
  - ▶ Visual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

- 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**

- 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](https://colab.research.google.com)

- **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](https://colab.research.google.com)
- **Mas você pode usar qualquer outro**

# Configurando o ambiente



The screenshot shows the Google Colab website interface. At the top, there's a navigation bar with the Google Colab logo and the text "Olá, este é o Colaboratory". Below this, there are menu items like "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". A search bar is present with the text "Comandos" and options for "+ Código" and "+ Texto".

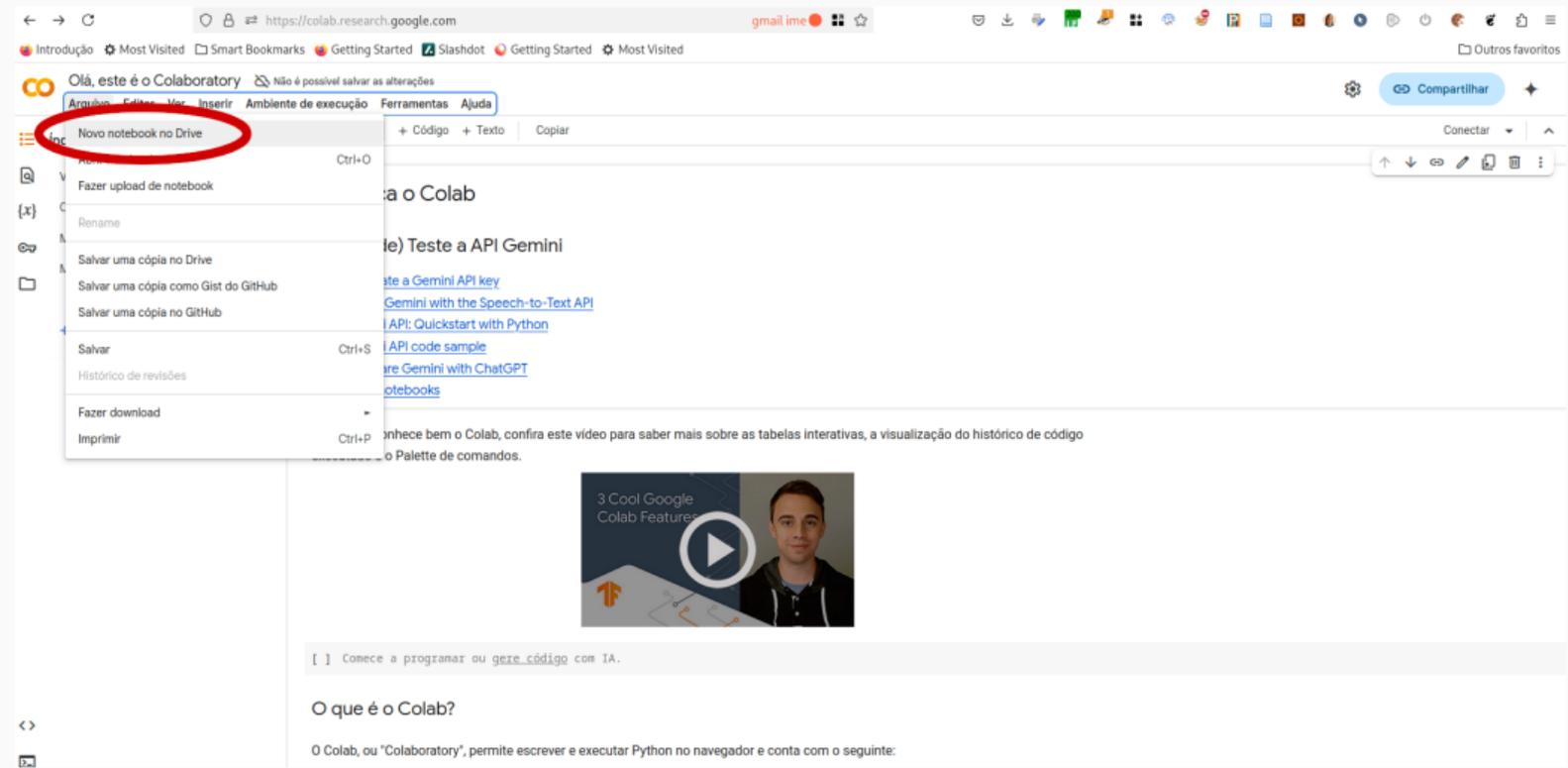
The main content area is titled "Conheça o Colab" and features a sub-heading "(Novidade) Teste a API Gemini". Below this, there is a list of links:

- [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](#)

Below the list, there is a paragraph: "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." This is followed by a video player thumbnail with the text "3 Cool Google Colab Features" and a play button icon.

At the bottom of the visible content, there is a section titled "O que é o Colab?" with the text: "O Colab, ou 'Colaboratory', permite escrever e executar Python no navegador e conta com o seguinte:"

# Configurando o ambiente



The screenshot shows the Google Colab web interface. The browser address bar displays `https://colab.research.google.com`. The main header area contains the text "Olá, este é o Colaboratory" and a notification "Não é possível salvar as alterações". Below this, a menu is open, and the option "Novo notebook no Drive" is highlighted with a red circle. Other menu items include "Fazer upload de notebook", "Rename", "Salvar uma cópia no Drive", "Salvar uma cópia como Gist do GitHub", "Salvar uma cópia no GitHub", "Salvar", "Histórico de revisões", "Fazer download", and "Imprimir". The main content area features a video player with the title "3 Cool Google Colab Features" and a play button. Below the video, there is a code input field with the text "[ ] Conecte a programar ou gere código com IA." and a heading "O que é o Colab?". The text below the heading reads: "O Colab, ou 'Colaboratory', permite escrever e executar Python no navegador e conta com o seguinte:".

# Configurando o ambiente

The image shows a web browser window displaying the Google Colab interface. The address bar shows the URL: <https://colab.research.google.com/drive/1MH891YA43iFWZ1pmmM-8ORXsrtiWG08l>. The page title is "Untitled0.ipynb". The main menu includes "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". The "Ambiente de execução" menu is open, showing a list of options with their corresponding keyboard shortcuts. The option "Alterar o tipo de ambiente de execução" is highlighted with a red circle. Other options include "Executar tudo" (Ctrl+F9), "Executar antes" (Ctrl+F8), "Executar a célula em foco" (Ctrl+Enter), "Executar seleção" (Ctrl+Shift+Enter), "Executar célula e abaixo" (Ctrl+F10), "Interromper execução" (Ctrl+M I), "Reiniciar sessão" (Ctrl+M .), "Reiniciar sessão e executar tudo", "Desconectar e atualizar ambiente de execução", "Gerenciar recursos", "Ver recursos", and "Ver registros do ambiente de execução".

Opção	Atalho
Executar tudo	Ctrl+F9
Executar antes	Ctrl+F8
Executar a célula em foco	Ctrl+Enter
Executar seleção	Ctrl+Shift+Enter
Executar célula e abaixo	Ctrl+F10
Interromper execução	Ctrl+M I
Reiniciar sessão	Ctrl+M .
Reiniciar sessão e executar tudo	
Desconectar e atualizar ambiente de execução	
<b>Alterar o tipo de ambiente de execução</b>	
Gerenciar recursos	
Ver recursos	
Ver registros do ambiente de execução	

# Configurando o ambiente

The image shows a Google Colab interface with a settings dialog box open. The dialog box is titled "Configurações" and has a sidebar with categories: Site, Editor, Assistência de IA, Colab Pro, GitHub, and Diversos. The "Editor" section is expanded, showing several settings: "Vinculações de teclas do editor" (default), "Tamanho da fonte (px)" (14), "Família de fontes usada para renderizar código" (monospace), "Largura do recuo em espaços" (2), and "Coluna com régua vertical" (80). In the "Diversos" section, three checkboxes are visible: "Mostrar sugestões de preenchimento de código com base no contexto" (checked and highlighted with a red circle), "Mostrar números de linha", and "Mostrar guias de recuo". At the bottom of the dialog are "Cancelar" and "Salvar" buttons.

Configurações

Site >

Vinculações de teclas do editor  
default

Editor >

Tamanho da fonte (px)  
14

Assistência de IA >

Família de fontes usada para renderizar código  
monospace

Colab Pro >

GitHub >

Largura do recuo em espaços  
2

Diversos >

Coluna com régua vertical  
80

Mostrar sugestões de preenchimento de código com base no contexto

Mostrar números de linha

Mostrar guias de recuo

Cancelar Salvar

# Configurando o ambiente

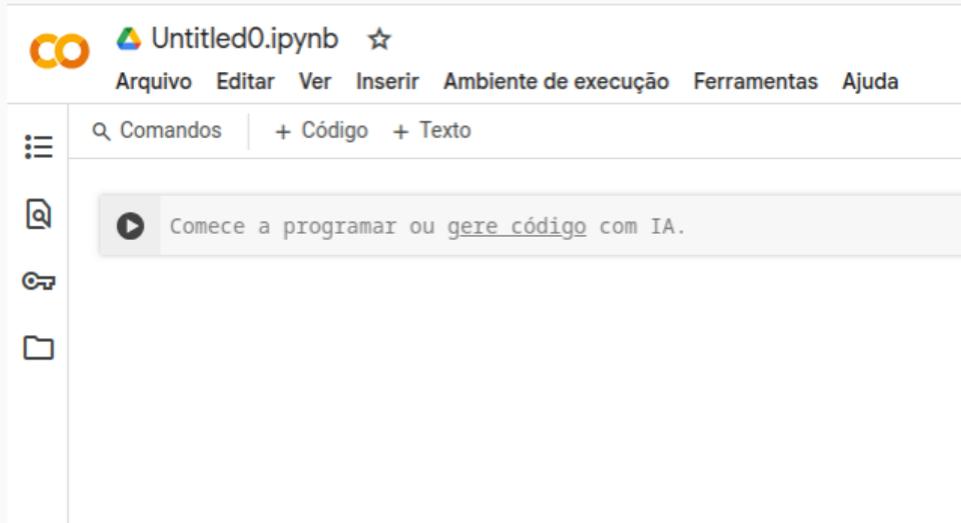
The image shows a Google Colab interface with a settings dialog box open. The dialog is titled "Configurações" and has a sidebar with categories: Site, Editor, Assistência de IA, Colab Pro, GitHub, and Diversos. Under the "Site" category, the option "Mostrar sugestões inline geradas com IA" is checked and circled in red. Other options include "Consentimento para usar recursos de IA generativa" and "Ocultar recursos de IA generativa". A disclaimer at the bottom of the dialog states: "A IA generativa no Colab é uma tecnologia experimental e pode, às vezes, mostrar informações imprecisas ou ofensivas que não representam as opiniões do Google. Portanto, [use o código com cuidado.](#)"

Configurações

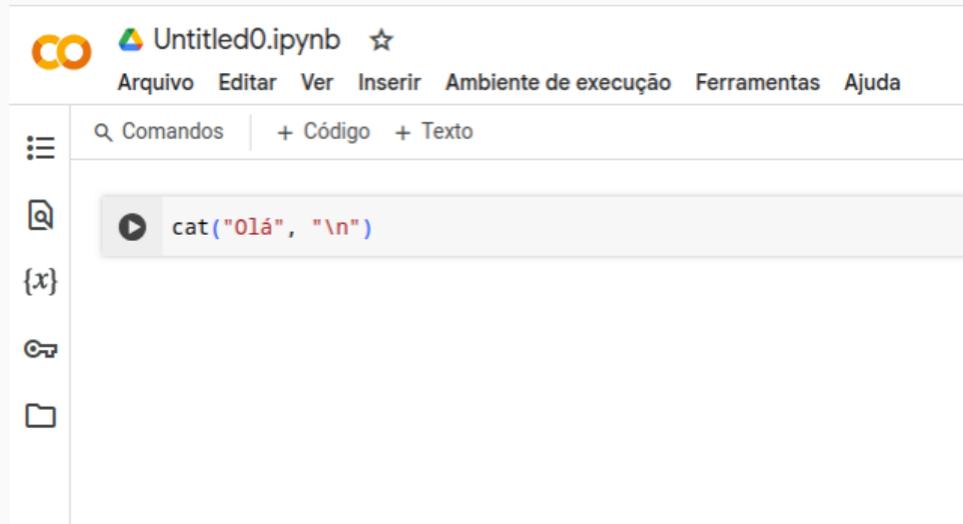
- Site >  Mostrar sugestões inline geradas com IA
- Editor >  Consentimento para usar recursos de IA generativa
- Assistência de IA >  Ocultar recursos de IA generativa
- Colab Pro >
- GitHub >
- Diversos >

A IA generativa no Colab é uma tecnologia experimental e pode, às vezes, mostrar informações imprecisas ou ofensivas que não representam as opiniões do Google. Portanto, [use o código com cuidado.](#)

Fechar

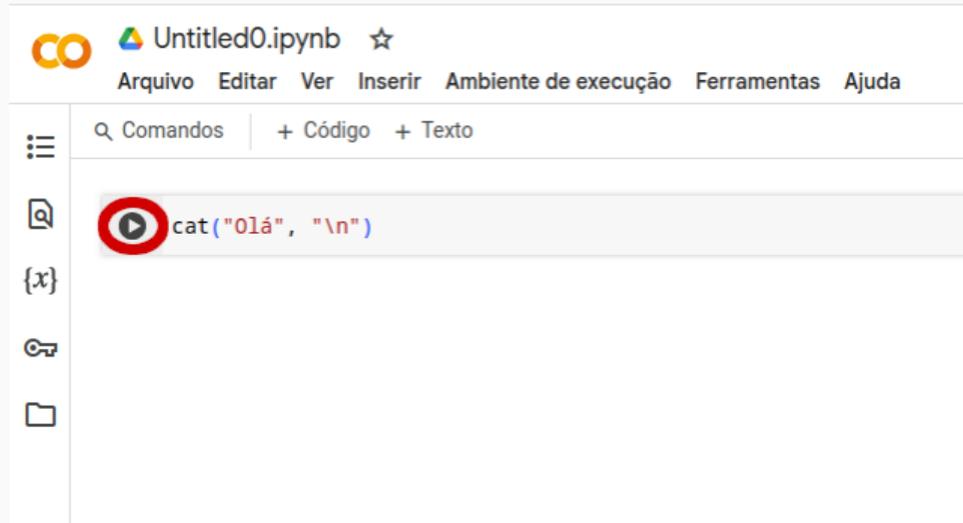


# cat()



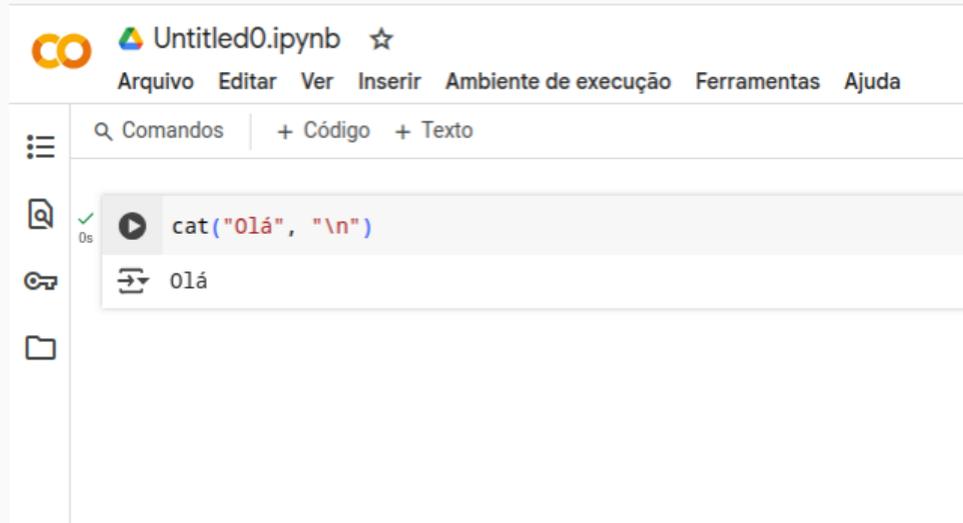
The screenshot shows a Jupyter Notebook window titled "Untitled0.ipynb". The interface includes a top menu bar with options: "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". Below the menu is a toolbar with "Comandos", "+ Código", and "+ Texto". A left sidebar contains icons for a menu, a notebook icon, a variable "{x}", a key icon, and a folder icon. The main area displays a code cell with a play button icon and the code `cat("Olá", "\n")`.

# cat()



The screenshot shows a Jupyter Notebook window titled "Untitled0.ipynb". The interface includes a top menu bar with options: "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". Below the menu is a toolbar with "Comandos", "Código", and "Texto" options. The main area contains a single code cell with the following Python code: `cat("Olá", "\n")`. The code is displayed in a light gray background with a red play button icon on the left side of the cell.

# cat()



The screenshot shows a Jupyter Notebook window titled "Untitled0.ipynb". The menu bar includes "Arquivo", "Editar", "Ver", "Inserir", "Ambiente de execução", "Ferramentas", and "Ajuda". The interface has a sidebar on the left with icons for home, search, and file management. The main area shows a code cell with the following content:

```
cat("Olá", "\n")
```

The cell is executed, indicated by a play button icon and a checkmark. The output of the cell is:

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



$$2 + 3 + 7$$

$$2 + 3 + 7$$

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

$$2 + 3 + 7$$

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

---

```
12
```

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

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

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

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

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

```
6.166667
```

Oops!

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

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

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

```
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) 18/30

# 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

```
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

```
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

```
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

```
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

```
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
0.3333333333333333
```

# 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
0.3333333333333333
```



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

- **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 "+")

- **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)

- **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?**

- **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*

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

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

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

- **Existem *tipos de dados* diferentes em R**

- ▶ Números inteiros, como 2 ou -437

- » *Não são comumente usados em R*

- » *Números entre  $\sim -2$  bilhões e  $\sim 2$  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")
```

```
5.000000000000000089
```

- **Existem *tipos de dados* diferentes em R**

- ▶ Números inteiros, como 2 ou -437

- » *Não são comumente usados em R*

- » *Números entre  $\sim -2$  bilhões e  $\sim 2$  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")
```

```
5.000000000000000089  Ok, números irracionais...
```

- **Existem *tipos de dados* diferentes em R**

- ▶ Números inteiros, como 2 ou -437

- » *Não são comumente usados em R*

- » *Números entre  $\sim -2$  bilhões e  $\sim 2$  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")
```

```
5.000000000000000089  Ok, números irracionais...
```

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

```
0.300000000000000044
```

- **Existem *tipos de dados* diferentes em R**

- ▶ Números inteiros, como 2 ou -437

- » *Não são comumente usados em R*

- » *Números entre  $\sim -2$  bilhões e  $\sim 2$  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")
```

```
5.000000000000000089    Ok, números irracionais...
```

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

```
0.300000000000000044    🤪
```

- **Existem *tipos de dados* diferentes em R**

- ▶ Números inteiros, como 2 ou -437

- » *Não são comumente usados em R*

- » *Números entre  $\sim -2$  bilhões e  $\sim 2$  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")`

- 5.0000000000000000089 *Ok, números irracionais...*

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

- 0.3000000000000000044 🤪

- ▶ 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  $\sim -2$  bilhões e  $\sim 2$  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")
```

```
5.000000000000000089 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)

$x \leftarrow 5$  (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)
```

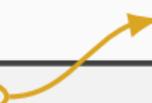
Há um número finito de caracteres no teclado, então fazemos atribuição em R com “<-” :

```
x <- 5
```

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

```
x = x + 1
```

expressão (valor)



**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:

```
x = x + 1
```

expressão (valor)

atribuição

**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:

```
x = x + 1
```

expressão (valor)

atribuição

nome

**AAAAAHHHH!!!!!!**

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

numeric

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

numeric

character

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

numeric

character

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

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

numeric

character

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

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
cat(class(x), "\n")
```

---

numeric

character

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

numeric

# Nomes (variáveis)

```
cat(class(5), "\n")
cat(class("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
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("0lar!"), "\n")
cat(class(x), "\n")
x <- 5
cat(class(x), "\n")
cat(class(5L), "\n")
x <- "0lar!"
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)

```
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)

```
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)

```
# 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

- **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

```
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

```
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 <- 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

```
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

```
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")
```