PyBR - Python em Português

Maxwell Anderson Ielpo do Amaral

Tutorial Completo para Iniciantes

Aprenda programação em Python usando português

Janeiro/2026

PyBR - Python Brasileiro
https://github.com/maxwellamaral/pybr

Sumário

Aprenda a Programar com PyBR - Guia Completo para Iniciantes

Autor: Maxwell Anderson Ielpo do Amaral

Publicado em Janeiro de 2026

Bem-vindo ao Mundo da Programação! 🚀

Este guia foi criado especialmente para você que nunca programou antes e quer aprender de forma fácil e em português! Com o **PyBR**, você vai aprender a programar usando palavras em português ao invés do inglês tradicional do Python.

Índice

7. Cálculos e Operações Matemáticas
8. Entrada e Saída de Dados
9. Tomando Decisões - Estruturas Condicionais
10. Repetindo Ações - Laços de Repetição
11. Organizando o Código - Funções
12. Criando Objetos - Classes
13. Projetos Práticos

Usando o Terminal - Guia para Iniciantes

Se você nunca usou o **Terminal** (também chamado de **Linha de Comando** ou **Prompt de Comando**), não se preocupe! É mais simples do que parece.

O que é o Terminal?

O Terminal é uma interface de texto onde você digita comandos para o computador executar. É como conversar com o computador através de texto ao invés de clicar com o mouse.

Por que usar? Programadores usam o Terminal porque é rápido, poderoso e permite automatizar tarefas!

No Windows

Como Abrir o Terminal no Windows: **Opção 1: Pelo Menu Iniciar** 1. Clique no botão **Iniciar** (ícone do Windows) 2. Digite `cmd` ou `powershell` 3. Pressione **Enter**

Opção 2: Atalho de Teclado 1. Pressione `Windows + R` 2. Digite `cmd` ou `powershell` 3. Pressione **Enter**

Opção 3: No VS Code 1. Abra o VS Code 2. Pressione `Ctrl + ` (acento grave) 3. O terminal aparecerá na parte inferior

Comandos Básicos no Windows:

```
# Ver onde você está (diretório atual)
cd

# Listar arquivos e pastas
dir

# Entrar em uma pasta
cd nome_da_pasta

# Voltar uma pasta acima
cd ..

# Ir para uma pasta específica (exemplo)
cd C:\Users\SeuNome\Downloads
```

```
# Limpar a tela
cls

# Ver conteúdo de um arquivo
type arquivo.txt
```

Navegando até a Pasta do PyBR (Exemplo no Windows):

```
# Se você salvou na pasta Downloads
cd C:\Users\SeuNome\Downloads\pybr

# Ou se está no Desktop
cd C:\Users\SeuNome\Desktop\pybr

# Verificar se está na pasta certa (deve listar pybr.py)
dir
```

No Mac/Linux

Como Abrir o Terminal no Mac: **Opção 1: Spotlight** 1. Pressione `Command + Espaço` 2. Digite `terminal` 3. Pressione **Enter**

Opção 2: Finder 1. Abra **Finder** 2. Vá em **Aplicativos → Utilitários → Terminal**

Como Abrir o Terminal no Linux: **Opção 1: Atalho de Teclado** - Pressione `Ctrl + Alt + T`

Opção 2: Menu de Aplicativos - Procure por "Terminal" no menu de aplicativos

Comandos Básicos no Mac/Linux:

```
# Ver onde você está (diretório atual)
pwd

# Listar arquivos e pastas
ls

# Listar com detalhes
ls -la

# Entrar em uma pasta
cd nome_da_pasta

# Voltar uma pasta acima
cd ..

# Ir para sua pasta pessoal
cd ~
```

```
# Ir para uma pasta específica (exemplo)
cd ~/Downloads/pybr

# Limpar a tela
clear

# Ver conteúdo de um arquivo
cat arquivo.txt
```

Navegando até a Pasta do PyBR (Exemplo no Mac/Linux):

```
# Se você salvou na pasta Downloads
cd ~/Downloads/pybr

# Ou se está no Desktop
cd ~/Desktop/pybr

# Verificar se está na pasta certa (deve listar pybr.py)
ls
```

Dicas Importantes para Usar o Terminal

1. Copiar e Colar no Terminal Windows (CMD): - Copiar: Selecione o texto e pressione `Enter` - Colar: Clique com botão direito

Windows (PowerShell) e Mac/Linux: - Copiar: `Ctrl + C` (Windows) ou `Command + C` (Mac) - Colar: `Ctrl + V` (Windows) ou `Command + V` (Mac) - No Linux: `Ctrl + Shift + C` e `Ctrl + Shift + V`

2. Autocompletar com TAB Digite o início de um nome de arquivo ou pasta e pressione **TAB** para completar automaticamente!

```
# Digite:
cd Doc[TAB]

# Completa para:
cd Documents
```

3. Histórico de Comandos Use as **setas ↑ ↓** do teclado para navegar pelos comandos que você já digitou.

4. Cancelar um Comando Se um programa travou ou você quer parar a execução: - Pressione `Ctrl + C`

5. Caminho Absoluto vs Relativo **Caminho Absoluto** - Especifica o caminho completo desde a raiz:

```
# Windows
C:\Users\SeuNome\pybr\exercicios\01-ola-mundo.pybr

# Mac/Linux
/Users/SeuNome/pybr/exercicios/01-ola-mundo.pybr
```

Caminho Relativo - Relativo à pasta atual:

```
# Se você já está na pasta pybr
exercicios/01-ola-mundo.pybr

# Ou com ./ (mesma coisa)
./exercicios/01-ola-mundo.pybr
```

Executando Seu Primeiro Comando PyBR

Agora que você sabe usar o Terminal, vamos executar um programa PyBR!

Passo a passo completo:

```
# 1. Navegue até a pasta do PyBR (ajuste o caminho conforme necessário)
cd caminho/para/pybr

# 2. Verifique se está no lugar certo
# Windows:
dir
# Mac/Linux:
ls

# Você deve ver: pybr.py, exercicios/, etc.

# 3. Execute seu primeiro programa!
python pybr.py exercicios/01-ola-mundo.pybr

# 4. Ou inicie o modo interativo
python pybr.py
```

Resultado esperado:

```
Olá, Mundo!
Meu nome é João
Estou aprendendo a programar!
PyBR é demais!
```

Nota: Se o comando python não funcionar, você precisa instalar o Python primeiro! Veja a próxima seção.

Problemas Comuns e Soluções

"python não é reconhecido como comando" **Solução:** Python não está instalado ou não está no PATH. Veja a próxima seção **"Instalando o Python"** para resolver isso!

"Não encontrou o arquivo pybr.py" **Solução:** Você não está na pasta correta.

1. Use `cd` para navegar até a pasta onde está o PyBR
2. Use `dir` (Windows) ou `ls` (Mac/Linux) para confirmar que vê o arquivo `pybr.py`

"Permissão negada" (Mac/Linux) **Solução:** Alguns arquivos precisam de permissão de execução.

```
# Dê permissão de execução
chmod +x pybr.py
```

Resumo - Comandos Essenciais

Ação	Windows	Mac/Linux
Onde estou?	`cd`	`pwd`
Listar arquivos	`dir`	`ls`
Entrar em pasta	`cd pasta`	`cd pasta`
Voltar	`cd ..`	`cd ..`
Limpar tela	`cls`	`clear`
Executar PyBR	`python pybr.py arquivo.pybr`	`python pybr.py arquivo.pybr`

Pronto! Agora você sabe usar o Terminal e está pronto para instalar o Python!

Instalando o Python

Antes de começar a programar com PyBR, você precisa ter o **Python** instalado no seu computador. O Python é a linguagem de programação que o PyBR traduz!

Verificando se o Python já está instalado

Primeiro, vamos verificar se você já tem o Python instalado:

Abra o Terminal (que você aprendeu na seção anterior) e digite:

```
python --version
```

Ou tente:

```
python3 --version
```

Ou no Windows:

```
py --version
```

Se aparecer algo como `Python 3.11.5` ou `Python 3.x.x`, **parabéns!** Você já tem o Python instalado e pode pular para a próxima seção.

Se aparecer uma mensagem de erro como "comando não encontrado" ou "não é reconhecido", continue lendo para instalar.

Instalando no Windows

Passo 1: Baixar o Python

1. Acesse o site oficial: https://www.python.org/downloads/
2. Clique no botão grande amarelo **"Download Python 3.x.x"**
3. O download do instalador começará automaticamente

Passo 2: Executar o Instalador

1. Abra o arquivo baixado (normalmente está na pasta **Downloads**)
2. ⚠ **IMPORTANTE:** Marque a caixa **"Add Python to PATH"** no início da instalação
   - Essa opção é ESSENCIAL para usar o Python no terminal!
3. Clique em **"Install Now"**
4. Aguarde a instalação (pode demorar alguns minutos)
5. Clique em **"Close"** quando terminar

Passo 3: Verificar a Instalação Abra um **NOVO** Terminal (feche o anterior se estiver aberto) e digite:

```
python --version
```

Deve aparecer a versão do Python instalada, exemplo: `Python 3.11.5`

Se não funcionar, tente:

```
py --version
```

Instalando no Mac

Opção 1: Usando o Site Oficial (Recomendado) Passo 1: Baixar o Python

1. Acesse: https://www.python.org/downloads/
2. Clique em **"Download Python 3.x.x"** para Mac
3. Baixe o instalador `.pkg`

Passo 2: Instalar

1. Abra o arquivo `.pkg` baixado
2. Siga o assistente de instalação:
   - Clique em **"Continue"** nas telas iniciais
   - Aceite a licença
   - Escolha o local de instalação (deixe o padrão)
   - Clique em **"Install"**
3. Digite sua senha de administrador quando solicitado
4. Clique em **"Close"** quando terminar

Passo 3: Verificar

Abra o Terminal e digite:

```
python3 --version
```

No Mac, geralmente usamos `python3` ao invés de `python`.

Opção 2: Usando Homebrew (Para Usuários Avançados) Se você já usa o Homebrew:

```
brew install python3
```

Instalando no Linux

A maioria das distribuições Linux já vem com Python instalado. Mas se precisar instalar ou atualizar:

Ubuntu/Debian

```
# Atualizar lista de pacotes
sudo apt update

# Instalar Python 3
sudo apt install python3 python3-pip

# Verificar instalação
python3 --version
```

Fedora

```
# Instalar Python 3
sudo dnf install python3 python3-pip

# Verificar instalação
python3 --version
```

Arch Linux

```
# Instalar Python 3
sudo pacman -S python python-pip

# Verificar instalação
python --version
```

Testando a Instalação Completa

Agora vamos testar se tudo está funcionando corretamente!

Teste 1: Versão do Python

```
# Windows
python --version

# Mac/Linux
python3 --version
```

Resultado esperado: Python 3.x.x (qualquer versão 3.6 ou superior)

Teste 2: Executar Python Interativo

```
# Windows
python

# Mac/Linux
python3
```

Você deve ver algo assim:

```
Python 3.11.5 (tags/v3.11.5:..., Aug  7 2023, 10:30:00)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Digite `exit()` e pressione Enter para sair.

Teste 3: Executar um Comando Python

```
# Windows
python -c "print('Python funcionando!')"

# Mac/Linux
python3 -c "print('Python funcionando!')"
```

Resultado esperado: `Python funcionando!`

Configurando Aliases (Opcional - Mac/Linux)

No Mac e Linux, é comum ter que digitar `python3` ao invés de `python`. Para facilitar, você pode criar um alias:

Bash (padrão no Ubuntu):

```
echo "alias python=python3" >> ~/.bashrc
source ~/.bashrc
```

Zsh (padrão no Mac moderno):

```
echo "alias python=python3" >> ~/.zshrc
source ~/.zshrc
```

Agora você pode usar apenas `python` ao invés de `python3`!

Problemas Comuns e Soluções

Windows: "Python não é reconhecido como comando" Causa: Python não foi adicionado ao PATH durante a instalação.

Solução 1 - Reinstalar: 1. Desinstale o Python pelo Painel de Controle 2. Reinstale marcando **"Add Python to PATH"**

Solução 2 - Adicionar manualmente ao PATH: 1. Procure onde o Python foi instalado (geralmente `C:\Users\SeuNome\AppData\Local\Programs\Python\Python3XX`) 2. Adicione esse caminho às variáveis de ambiente do Windows 3. Tutorial: Adicionar ao PATH no Windows

Mac: "Python 2.x aparece ao invés de Python 3" Causa: Mac vem com Python 2 pré-instalado.

Solução: Sempre use `python3` ao invés de `python`, ou configure um alias.

Linux: "Permissão negada" Causa: Alguns comandos precisam de privilégios de administrador.

Solução: Use sudo antes do comando:

```
sudo apt install python3
```

Erro: "pip não encontrado" Solução: Instale o pip (gerenciador de pacotes Python):

```
# Windows
python -m ensurepip --upgrade

# Mac/Linux
python3 -m ensurepip --upgrade
```

Resumo - Comandos Python Essenciais

Comando	Windows	Mac/Linux
Verificar versão	`python --version`	`python3 --version`
Abrir Python interativo	`python`	`python3`
Executar arquivo	`python arquivo.py`	`python3 arquivo.py`
Instalar pacote	`pip install pacote`	`pip3 install pacote`
Sair do Python	`exit()`	`exit()`

Perfeito! Agora você tem o Python instalado e testado, está pronto para usar o PyBR!

Como Executar o PyBR

Agora que você tem o Python instalado e sabe usar o Terminal, está pronto para executar programas PyBR!

O que você precisa

 Python 3.6 ou superior - Você já instalou na seção anterior!
 Arquivos do PyBR - O transpiler `pybr.py` e os exemplos
 Terminal aberto - Para executar os comandos

Baixando o PyBR

1. Baixe ou clone o projeto PyBR do repositório
2. Navegue até a pasta do projeto no terminal:

   ```
   cd caminho/para/pasta/pybr
   ```

Formas de Executar Código PyBR

Opção 1: Modo Interativo (REPL) - Recomendado para Iniciantes O REPL é perfeito para **testar comandos rapidamente** e experimentar os exemplos deste tutorial!

Para iniciar o modo interativo, digite no terminal:

```
python pybr.py
```

Você verá algo assim:

```
PyBR - Python em Português
Digite 'sair()' para encerrar
>>>
```

Agora você pode digitar comandos diretamente:

```
>>> imprimir("Olá, Mundo!")
Olá, Mundo!
>>> x = 10
>>> imprimir(x * 2)
20
>>> sair()
```

 Dica: Use o REPL para testar cada exemplo pequeno deste tutorial!

Opção 2: Criar e Executar Arquivos .pybr Para programas maiores, crie um arquivo de texto com a extensão `.pybr`:

Passo 1: Crie um arquivo chamado `meu_programa.pybr` (pode usar qualquer editor de texto ou VS Code)

Passo 2: Escreva seu código PyBR no arquivo:

```
# meu_programa.pybr
imprimir("Meu primeiro programa!")

nome = entrada("Qual é seu nome? ")
imprimir(f"Olá, {nome}!")
```

Passo 3: Execute o arquivo no terminal:

```
python pybr.py meu_programa.pybr
```

Opção 3: Usar o VS Code com Syntax Highlighting Para uma melhor experiência de desenvolvimento:

1. Instale o Visual Studio Code
2. Instale a extensão PyBR (veja instruções no README.md principal)
3. Crie arquivos `.pybr` com destaque de sintaxe colorido
4. Execute pelo terminal integrado do VS Code

Como Usar Este Tutorial

Para cada exemplo neste tutorial, você pode:

1. **Exemplos curtos (1-3 linhas):** Digite no REPL interativo

   ```
   >>> imprimir("Testando!")
   ```

2. **Executar os arquivos prontos:** Use os arquivos `.pybr` da pasta `exercicios/`

   ```
   python pybr.py exercicios/01-ola-mundo.pybr
   ```

3. **Exemplos médios:** Copie e cole no REPL (algumas linhas por vez)
4. **Exemplos longos e projetos:** Crie um arquivo `.pybr`, cole o código e execute

Testando Sua Instalação

Vamos testar se tudo está funcionando! Execute este código:

No REPL:

```
>>> imprimir("PyBR funcionando!")
>>> para i em intervalo(3):
...     imprimir(f"Contagem: {i}")
...
```

Resultado esperado:

```
PyBR funcionando!
Contagem: 0
Contagem: 1
Contagem: 2
```

Se você viu essa saída, está tudo pronto!

Arquivos de Exemplo Prontos

Todos os exemplos deste tutorial estão disponíveis como arquivos `.pybr` na pasta `exercicios/`. Você pode executá-los diretamente:

```
# Exemplo: executar o primeiro programa
python pybr.py exercicios/01-ola-mundo.pybr

# Exemplo: executar a calculadora de IMC
python pybr.py exercicios/05-calculadora-imc.pybr

# Exemplo: executar o jogo de adivinhação
python pybr.py exercicios/11-jogo-adivinhacao.pybr
```

Dica: Olhe o ícone no início de cada seção para saber qual arquivo corresponde àquele conteúdo!

O que é Programação?

Programar é dar instruções ao computador, como se você estivesse escrevendo uma receita de bolo! Assim como em uma receita você diz "misture os ingredientes", "asse por 30 minutos", na programação você diz ao computador "calcule isso", "mostre aquilo", "repita esta ação".

O computador é muito rápido, mas precisa de instruções **muito detalhadas**. Ele faz exatamente o que você mandar - nem mais, nem menos!

Seu Primeiro Programa

 Arquivo de exemplo: `exercicios/01-ola-mundo.pybr`

Vamos começar com o clássico "Olá, Mundo!":

```
imprimir("Olá, Mundo!")
```

O que acontece aqui? - `imprimir()` é uma **função** que mostra texto na tela - O texto entre aspas `"Olá, Mundo!"` é o que será mostrado - As aspas dizem ao computador: "isso é texto, não é código"

Experimente você mesmo:

```
imprimir("Meu nome é João")
imprimir("Estou aprendendo a programar!")
imprimir("PyBR é demais!")
```

Cada `imprimir()` mostra uma linha diferente na tela.

Variáveis - A Memória do Computador

 Arquivo de exemplo: `exercicios/02-variaveis.pybr`

O que são Variáveis?

Imagine que o computador tem milhares de caixinhas onde pode guardar informações. As **variáveis** são como etiquetas que você cola nessas caixinhas para lembrar o que tem dentro.

Por que existem? - Para **guardar** informações que você vai usar depois - Para **reutilizar** valores sem ter que digitá-los novamente - Para fazer o programa **lembrar** de coisas

Como criar variáveis:

```
# Guardando um nome
nome = "Maria"
```

```
# Guardando uma idade
idade = 25

# Guardando um preço
preco = 19.90

# Usando as variáveis
imprimir(nome)
imprimir(idade)
imprimir(preco)
```

Explicando: - nome é a etiqueta da caixinha - = significa “guarde nesta caixinha” - "Maria" é o valor que vai ser guardado

Analogia do Mundo Real:

Pense nas variáveis como gavetas etiquetadas: - **Gaveta “nome”**: contém “Maria” - **Gaveta “idade”**: contém 25 - **Gaveta “preco”**: contém 19.90

Tipos de Dados:

```
# TEXTO (chamamos de "string")
cidade = "São Paulo"
mensagem = "Bem-vindo!"

# NÚMEROS INTEIROS
quantidade = 10
ano = 2026

# NÚMEROS DECIMAIS
altura = 1.75
temperatura = 23.5

# VERDADEIRO ou FALSO (chamamos de "booleano")
esta_chovendo = Falso
esta_ensolarado = Verdadeiro
```

Mudando o valor de uma variável:

```
saldo = 100
imprimir(saldo)  # Mostra: 100

saldo = 150
imprimir(saldo)  # Mostra: 150

saldo = saldo + 50
imprimir(saldo)  # Mostra: 200
```

Cálculos e Operações Matemáticas

 Arquivo de exemplo: `exercicios/03-calculos.pybr`

O computador é uma super calculadora! Veja o que você pode fazer:

Operações Básicas:

```
# ADIÇÃO (+)
soma = 10 + 5
imprimir(soma)  # Mostra: 15

# SUBTRAÇÃO (-)
diferenca = 20 - 8
imprimir(diferenca)  # Mostra: 12

# MULTIPLICAÇÃO (*)
produto = 6 * 7
imprimir(produto)  # Mostra: 42

# DIVISÃO (/)
resultado = 15 / 3
imprimir(resultado)  # Mostra: 5.0

# DIVISÃO INTEIRA (//)
resultado_inteiro = 17 // 5
imprimir(resultado_inteiro)  # Mostra: 3

# RESTO DA DIVISÃO (%)
resto = 17 % 5
imprimir(resto)  # Mostra: 2

# POTÊNCIA (**)
potencia = 2 ** 3
imprimir(potencia)  # Mostra: 8 (2 elevado a 3)
```

Calculadora de Compras:

```
# Preços dos produtos
preco_arroz = 25.90
preco_feijao = 8.50
preco_acucar = 4.20

# Calculando o total
total = preco_arroz + preco_feijao + preco_acucar
imprimir("Total da compra: R$")
imprimir(total)

# Calculando com desconto de 10%
desconto = total * 0.10
```

```
total_com_desconto = total - desconto
imprimir("Total com desconto: R$")
imprimir(total_com_desconto)
```

Ordem das Operações (como na matemática):

```
# O computador segue a mesma ordem da matemática
resultado = 2 + 3 * 4
imprimir(resultado)  # Mostra: 14 (primeiro 3*4, depois +2)

# Use parênteses para mudar a ordem
resultado = (2 + 3) * 4
imprimir(resultado)  # Mostra: 20 (primeiro 2+3, depois *4)
```

Entrada e Saída de Dados

 Arquivos de exemplo: exercicios/04-entrada-saida.pybr e exercicios/05-calculadora-imc.pybr

Saída - Mostrando Informações:

```
# Forma básica
imprimir("Olá!")

# Mostrando variáveis
nome = "Carlos"
imprimir(nome)

# Juntando texto e variáveis (f-strings)
idade = 30
imprimir(f"Meu nome é {nome} e tenho {idade} anos")

# Mostrando várias coisas na mesma linha
imprimir("A soma de 5 + 3 é:", 5 + 3)
```

Entrada - Recebendo Informações do Usuário:

```
# Pedindo o nome do usuário
nome = entrada("Digite seu nome: ")
imprimir(f"Olá, {nome}!")

# Pedindo a idade (lembre-se de converter para número)
idade_texto = entrada("Digite sua idade: ")
idade = inteiro(idade_texto)
imprimir(f"Você tem {idade} anos")

# Forma mais curta (convertendo direto)
idade = inteiro(entrada("Digite sua idade: "))
```

Programa Interativo Completo:

```
# Calculadora de IMC (Índice de Massa Corporal)
imprimir("=== Calculadora de IMC ===")

nome = entrada("Qual é seu nome? ")
peso = flutuante(entrada("Qual é seu peso em kg? "))
altura = flutuante(entrada("Qual é sua altura em metros? "))

imc = peso / (altura ** 2)

imprimir(f"\n{nome}, seu IMC é: {imc:.2f}")
```

Explicação: - `entrada()` sempre recebe texto - Para fazer cálculos, precisamos converter com `inteiro()` ou `flutuante()` - `{imc:.2f}` mostra o número com 2 casas decimais - `\n` cria uma linha em branco

---

Tomando Decisões - Estruturas Condicionais

 Arquivos de exemplo: `exercicios/06-condicionais.pybr` e `exercicios/07-sistema-login.pybr`

Programas precisam tomar decisões! É como um fluxograma: "SE isso acontecer, faça aquilo".

Estrutura SE (if):

```
idade = 18

se idade >= 18:
    imprimir("Você é maior de idade")
    imprimir("Pode tirar carteira de motorista")
```

Importante: - Depois do `:` você deve dar um **TAB** (identação) - Tudo que estiver identado só acontece se a condição for verdadeira

Estrutura SE-SENÃO (if-else):

```
idade = 15

se idade >= 18:
    imprimir("Você é maior de idade")
senao:
    imprimir("Você é menor de idade")
```

Estrutura SE-SENÃOSE-SENÃO (if-elif-else):

```
nota = 75

se nota >= 90:
    imprimir("Conceito: A - Excelente!")
```

```
senaose nota >= 70:
    imprimir("Conceito: B - Bom!")
senaose nota >= 50:
    imprimir("Conceito: C - Regular")
senao:
    imprimir("Conceito: D - Insuficiente")
```

Operadores de Comparação:

```
# == (igual a)
se 5 == 5:
    imprimir("São iguais")

# != (diferente de)
se 5 != 3:
    imprimir("São diferentes")

# > (maior que)
se 10 > 5:
    imprimir("10 é maior que 5")

# < (menor que)
se 3 < 7:
    imprimir("3 é menor que 7")

# >= (maior ou igual)
se 5 >= 5:
    imprimir("Verdadeiro")

# <= (menor ou igual)
se 4 <= 8:
    imprimir("Verdadeiro")
```

Operadores Lógicos:

```
# E (and) - ambas condições devem ser verdadeiras
idade = 20
tem_carteira = Verdadeiro

se idade >= 18 e tem_carteira:
    imprimir("Pode dirigir!")

# OU (or) - pelo menos uma condição deve ser verdadeira
dia = "sábado"

se dia == "sábado" ou dia == "domingo":
    imprimir("É fim de semana!")

# NÃO (not) - inverte a condição
```

```
chovendo = Falso

se nao chovendo:
    imprimir("Vamos ao parque!")
```

Exemplo Prático - Sistema de Login:

```
usuario_correto = "admin"
senha_correta = "1234"

usuario = entrada("Digite o usuário: ")
senha = entrada("Digite a senha: ")

se usuario == usuario_correto e senha == senha_correta:
    imprimir("✓ Login realizado com sucesso!")
    imprimir("Bem-vindo ao sistema!")
senao:
    imprimir("✗ Usuário ou senha incorretos!")
```

Repetindo Ações - Laços de Repetição

 Arquivos de exemplo: `exercicios/08-lacos-para.pybr`, `exercicios/09-tabuada.pybr`, `exercicios/10-enquanto.pybr`, `exercicios/11-jogo-adivinhacao.pybr` e `exercicios/12-listas.pybr`

Imagine ter que escrever `imprimir("Olá")` 100 vezes... Os laços de repetição fazem isso automaticamente!

Laço PARA (for) - Número Definido de Repetições:

```
# Contando de 0 a 4
para i em intervalo(5):
    imprimir(i)

# Resultado:
# 0
# 1
# 2
# 3
# 4
```

Explicação: - `para` inicia o laço - `i` é a variável que vai mudando (pode ter qualquer nome) - `em` indica onde buscar os valores - `intervalo(5)` gera números de 0 a 4

Personalizando o intervalo:

```
# De 1 a 10
para numero em intervalo(1, 11):
    imprimir(numero)
```

```
# De 0 a 10, pulando de 2 em 2
para numero em intervalo(0, 11, 2):
    imprimir(numero)  # Mostra: 0, 2, 4, 6, 8, 10

# Contagem regressiva
para numero em intervalo(10, 0, -1):
    imprimir(numero)  # Mostra: 10, 9, 8, ..., 1
```

Laço ENQUANTO (while) - Repete Enquanto Condição For Verdadeira:

```
contador = 0

enquanto contador < 5:
    imprimir(f"Contador: {contador}")
    contador = contador + 1

# Resultado:
# Contador: 0
# Contador: 1
# Contador: 2
# Contador: 3
# Contador: 4
```

Cuidado! Se a condição nunca ficar falsa, o programa fica preso em um loop infinito!

Exemplo Prático - Tabuada:

```
numero = inteiro(entrada("Digite um número para ver a tabuada: "))

imprimir(f"\n=== Tabuada do {numero} ===")
para i em intervalo(1, 11):
    resultado = numero * i
    imprimir(f"{numero} x {i} = {resultado}")
```

Exemplo Prático - Jogo de Adivinhação:

```
numero_secreto = 42
tentativas = 0

imprimir("=== Jogo de Adivinhação ===")
imprimir("Tente adivinhar o número entre 1 e 100!")

enquanto Verdadeiro:
    palpite = inteiro(entrada("\nSeu palpite: "))
    tentativas = tentativas + 1

    se palpite == numero_secreto:
        imprimir(f"□ Parabéns! Você acertou em {tentativas} tentativas!")
        quebre  # Sai do laço
```

```
    senaose palpite < numero_secreto:
        imprimir("□ O número secreto é MAIOR!")
    senao:
        imprimir("□ O número secreto é MENOR!")
```

Trabalhando com Listas:

```
# Criando uma lista
frutas = ["maçã", "banana", "laranja", "uva"]

# Percorrendo a lista
para fruta em frutas:
    imprimir(f"Eu gosto de {fruta}")

# Com índice (posição)
para i em intervalo(tamanho(frutas)):
    imprimir(f"{i + 1}. {frutas[i]}")
```

Funções Úteis para Listas e Números:

PyBR oferece várias funções prontas que facilitam muito o trabalho com listas e números!

Funções Matemáticas:

```
# Lista de números
numeros = [10, 5, 8, 3, 15, 2]

# Encontrar o maior valor
maior = maximo(numeros)
imprimir(f"Maior número: {maior}")  # 15

# Encontrar o menor valor
menor = minimo(numeros)
imprimir(f"Menor número: {menor}")  # 2

# Também funciona com múltiplos valores
imprimir(maximo(10, 25, 5, 30))  # 30
imprimir(minimo(10, 25, 5, 30))  # 5
```

Arredondamento e Potências:

```
# Arredondar números decimais
valor = 3.7
imprimir(arredondar(valor))  # 4

valor = 3.14159
total_arredondado = arredondar(valor, 2)
imprimir(total_arredondado)  # 3.14 (2 casas decimais)

# Potenciação (usar operador **)
```

```
resultado = 2 ** 3  # 2 elevado a 3
imprimir(resultado)  # 8

# Valor absoluto (sempre positivo)
imprimir(abs(-10))  # 10
imprimir(abs(10))   # 10
```

Ordenação e Reversão:

```
# Lista desordenada
numeros = [5, 2, 8, 1, 9, 3]

# Ordenar (do menor para o maior)
numeros_ordenados = ordenar(numeros)
imprimir(lista(numeros_ordenados))  # [1, 2, 3, 5, 8, 9]

# Reverter a ordem
numeros_invertidos = lista(reverter(numeros))
imprimir(numeros_invertidos)  # [3, 9, 1, 8, 2, 5]

# Funciona com textos também!
nomes = ["Carlos", "Ana", "Bruno"]
nomes_ordenados = ordenar(nomes)
imprimir(lista(nomes_ordenados))  # ['Ana', 'Bruno', 'Carlos']
```

Enumeração (índice + valor):

```
# Quando você precisa do índice E do valor
frutas = ["maçã", "banana", "uva"]

para indice, fruta em enumerar(frutas):
    imprimir(f"{indice + 1}º fruta: {fruta}")

# Saída:
# 1º fruta: maçã
# 2º fruta: banana
# 3º fruta: uva
```

Verificações com qualquer() e todos():

```
# Verificar se ALGUM valor é True
notas = [5, 7, 9, 6]
tem_nota_alta = qualquer([nota >= 9 para nota em notas])
imprimir(tem_nota_alta)  # Verdadeiro (existe uma nota 9)

# Verificar se TODOS os valores são True
todas_aprovadas = todos([nota >= 6 para nota em notas])
imprimir(todas_aprovadas)  # Verdadeiro (todas >= 6)
```

Exemplo Completo - Análise de Notas:

```
# Notas de um aluno
notas = [7.5, 8.0, 6.5, 9.0, 7.0]

# Análise completa
imprimir(f"Notas: {notas}")

maior_nota = maximo(notas)
imprimir(f"Maior nota: {maior_nota}")

menor_nota = minimo(notas)
imprimir(f"Menor nota: {menor_nota}")

total = sum(notas)
media = total / tamanho(notas)
media_arredondada = arredondar(media, 2)
imprimir(f"Média: {media_arredondada}")

# Verificar aprovação (média >= 7)
se media >= 7:
    imprimir("Aluno APROVADO! □")
senao:
    imprimir("Aluno em recuperação")

# Ordenar notas (da menor para maior)
notas_ordenadas = ordenar(notas)
imprimir(f"Notas em ordem: {lista(notas_ordenadas)}")
```

Organizando o Código - Funções

□ **Arquivos de exemplo:** `exercicios/13-funcoes-simples.pybr`, `exercicios/14-funcoes-retorno.pybr` e `exercicios/15-calculadora-funcoes.pybr`

Funções são como **receitas reutilizáveis**. Você define uma vez e pode usar várias vezes!

Por que usar funções?

1. **Evitar repetição** - Escreva uma vez, use muitas vezes
2. **Organização** - Código mais limpo e fácil de entender
3. **Facilitar manutenção** - Alterar em um lugar só

Criando uma Função Simples:

```
# Definindo a função
definir saudar():
    imprimir("Olá, seja bem-vindo!")
    imprimir("Tenha um ótimo dia!")
```

```
# Usando a função
saudar()
saudar()  # Podemos chamar quantas vezes quiser
```

Funções com Parâmetros (Argumentos):

```
# Função que recebe um nome
definir saudar_pessoa(nome):
    imprimir(f"Olá, {nome}!")
    imprimir("Como você está?")

# Usando
saudar_pessoa("Maria")
saudar_pessoa("João")
saudar_pessoa("Ana")
```

Funções com Múltiplos Parâmetros:

```
definir calcular_media(nota1, nota2, nota3):
    soma = nota1 + nota2 + nota3
    media = soma / 3
    imprimir(f"A média é: {media:.2f}")

calcular_media(7.5, 8.0, 9.5)
calcular_media(6.0, 7.0, 8.5)
```

Funções que RETORNAM Valores:

```
definir somar(a, b):
    resultado = a + b
    retornar resultado

# Usando o valor retornado
total = somar(10, 5)
imprimir(total)  # Mostra: 15

# Ou diretamente
imprimir(somar(20, 30))  # Mostra: 50
```

Exemplo Completo - Calculadora:

```
definir somar(a, b):
    retornar a + b

definir subtrair(a, b):
    retornar a - b

definir multiplicar(a, b):
    retornar a * b
```

```
definir dividir(a, b):
    se b == 0:
        retornar "Erro: Divisão por zero!"
    retornar a / b

# Menu da calculadora
imprimir("=== CALCULADORA ===")
imprimir("1. Somar")
imprimir("2. Subtrair")
imprimir("3. Multiplicar")
imprimir("4. Dividir")

opcao = inteiro(entrada("\nEscolha uma opção: "))
num1 = flutuante(entrada("Digite o primeiro número: "))
num2 = flutuante(entrada("Digite o segundo número: "))

se opcao == 1:
    imprimir(f"Resultado: {somar(num1, num2)}")
senaose opcao == 2:
    imprimir(f"Resultado: {subtrair(num1, num2)}")
senaose opcao == 3:
    imprimir(f"Resultado: {multiplicar(num1, num2)}")
senaose opcao == 4:
    imprimir(f"Resultado: {dividir(num1, num2)}")
senao:
    imprimir("Opção inválida!")
```

Funções com Valores Padrão:

```
definir fazer_cafe(tipo="normal", acucar=1):
    imprimir(f"Fazendo café {tipo} com {acucar} colher(es) de açúcar")

fazer_cafe()  # Usa valores padrão
fazer_cafe("expresso")  # Usa açúcar padrão
fazer_cafe("cappuccino", 2)  # Define tudo
fazer_cafe(acucar=0)  # Café sem açúcar
```

Funções Avançadas - Lambda, Filtrar e Mapear

 Arquivo de exemplo: `exercicios/23-funcoes-avancadas.pybr`

Agora vamos aprender sobre **ferramentas poderosas** para trabalhar com listas de forma eficiente!

Funções Lambda (Funções Anônimas):

Lambda são **funções pequenas** que você cria rapidamente, sem precisar usar `definir`.

```
# Função normal
definir dobro(x):
    retornar x * 2

# Mesma função usando lambda
dobro_lambda = lambda x: x * 2

# Usando
imprimir(dobro(5))        # 10
imprimir(dobro_lambda(5)) # 10
```

Quando usar lambda? - Para funções simples de uma linha - Quando você precisa de uma função rápida e não vai reutilizar muito

```
# Mais exemplos
quadrado = lambda x: x * x
eh_par = lambda n: n % 2 == 0
saudar = lambda nome: f"Olá, {nome}!"

imprimir(quadrado(4))      # 16
imprimir(eh_par(7))        # Falso
imprimir(saudar("Maria"))  # Olá, Maria!
```

Mapear - Aplicar Função a Todos Elementos:

A função `mapear()` aplica uma função a **cada elemento** de uma lista.

```
# Dobrar todos os números
numeros = [1, 2, 3, 4, 5]
dobrados = lista(mapear(lambda x: x * 2, numeros))
imprimir(dobrados)  # [2, 4, 6, 8, 10]

# Converter para maiúsculas
nomes = ["ana", "joão", "maria"]
nomes_maiusculos = lista(mapear(lambda n: n.upper(), nomes))
imprimir(nomes_maiusculos)  # ['ANA', 'JOÃO', 'MARIA']

# Calcular quadrados
numeros = [1, 2, 3, 4]
quadrados = lista(mapear(lambda x: x ** 2, numeros))
imprimir(quadrados)  # [1, 4, 9, 16]
```

Filtrar - Selecionar Elementos:

A função `filtrar()` **seleciona apenas os elementos** que atendem a uma condição.

```
# Filtrar números pares
numeros = [1, 2, 3, 4, 5, 6, 7, 8]
pares = lista(filtrar(lambda x: x % 2 == 0, numeros))
imprimir(pares)  # [2, 4, 6, 8]
```

```
# Filtrar números maiores que 5
maiores_que_5 = lista(filtrar(lambda x: x > 5, numeros))
imprimir(maiores_que_5)  # [6, 7, 8]

# Filtrar nomes com mais de 4 letras
nomes = ["Ana", "João", "Maria", "José"]
nomes_longos = lista(filtrar(lambda n: tamanho(n) > 4, nomes))
imprimir(nomes_longos)  # ['Maria']
```

Combinando Filtrar e Mapear:

Você pode **combinar** essas funções para fazer coisas incríveis!

```
# Pegar números pares e dobrar seus valores
numeros = [1, 2, 3, 4, 5, 6, 7, 8]

# Primeiro filtra os pares, depois dobra
pares = filtrar(lambda x: x % 2 == 0, numeros)
dobrados = lista(mapear(lambda x: x * 2, pares))
imprimir(dobrados)  # [4, 8, 12, 16]

# Ou em uma linha só:
resultado = lista(mapear(lambda x: x * 2, filtrar(lambda x: x % 2 == 0, numeros)))
imprimir(resultado)  # [4, 8, 12, 16]
```

Exemplo Prático - Processamento de Notas:

```
# Lista de notas
notas = [8.5, 5.0, 7.5, 4.0, 9.0, 6.5, 3.5, 8.0, 9.5, 5.5]
imprimir(f"Notas dos alunos: {notas}")

total_alunos = tamanho(notas)
imprimir(f"Total de alunos: {total_alunos}")

# Estatísticas gerais
maior_nota = maximo(notas)
imprimir(f"Maior nota: {maior_nota}")

menor_nota = minimo(notas)
imprimir(f"Menor nota: {menor_nota}")

media_turma = arredondar(sum(notas) / tamanho(notas), 2)
imprimir(f"Média da turma: {media_turma}")

# Filtrar aprovados (>= 6)
aprovados = lista(filtrar(lambda n: n >= 6, notas))
imprimir(f"Notas dos aprovados: {aprovados}")

total_aprovados = tamanho(aprovados)
imprimir(f"Total de aprovados: {total_aprovados}")
```

```
# Filtrar reprovados (< 6)
reprovados = lista(filtrar(lambda n: n < 6, notas))
imprimir(f"Notas dos reprovados: {reprovados}")

# Adicionar ponto extra de 0.5 para todos
com_bonus = lista(mapear(lambda n: minimo(n + 0.5, 10), notas))
imprimir(f"Com ponto extra: {com_bonus}")

# Verificar quantos aprovaram após bônus
aprovados_bonus = lista(filtrar(lambda n: n >= 6, com_bonus))
total_aprovados_bonus = tamanho(aprovados_bonus)
imprimir(f"Aprovados após bônus: {total_aprovados_bonus}")
```

Exemplo Prático - Lista de Produtos:

```
# Simulando produtos (usando listas paralelas)
precos = [15.50, 8.00, 22.00, 5.50, 35.00, 12.00]
nomes = ["Arroz", "Feijão", "Carne", "Macarrão", "Picanha", "Óleo"]

imprimir("--- Lista de preços ---")
para i em intervalo(tamanho(nomes)):
    imprimir(f"{nomes[i]}: R$ {precos[i]:.2f}")

# Filtrar produtos baratos (<= 15)
indices_baratos = lista(filtrar(
    lambda i: precos[i] <= 15,
    intervalo(tamanho(precos))
))

imprimir("\n--- Produtos baratos (até R$ 15) ---")
para i em indices_baratos:
    imprimir(f"{nomes[i]}: R$ {precos[i]:.2f}")

# Aplicar desconto de 10% em todos
precos_com_desconto = lista(mapear(lambda p: arredondar(p * 0.9, 2), precos))

imprimir("\n--- Preços com 10% de desconto ---")
para i em intervalo(tamanho(nomes)):
    imprimir(f"{nomes[i]}: R$ {precos_com_desconto[i]:.2f}")
```

Por que usar essas funções? - □ **Código mais limpo** - Menos linhas, mais expressivo - ⚡ **Mais eficiente** - Processamento otimizado - □ **Programação funcional** - Estilo moderno de programação - □ **Reutilizável** - Funções lambda podem ser passadas como parâmetros

---

Criando Objetos - Classes

 Arquivos de exemplo: `exercicios/16-classe-cachorro.pybr`, `exercicios/17-classe-conta-bancaria.pybr`, `exercicios/18-classe-retangulo.pybr` e `exercicios/19-classe-aluno.pybr`

Classes são como **moldes** para criar objetos. É como uma receita de bolo (classe) que você usa para fazer vários bolos (objetos).

O que são Objetos?

Objetos são "coisas" que têm: - **Características** (atributos) - o que o objeto é - **Comportamentos** (métodos) - o que o objeto faz

Exemplo do mundo real: - **Carro** (classe) - Características: cor, modelo, ano, velocidade - Comportamentos: acelerar, frear, buzinar

Criando Sua Primeira Classe:

```
classe Cachorro:
    # Método construtor - executado ao criar um cachorro
    definir __init__(proprio, nome, raca):
        proprio.nome = nome
        proprio.raca = raca

    # Método - comportamento do cachorro
    definir latir(proprio):
        imprimir(f"{proprio.nome}: Au au!")

    definir apresentar(proprio):
        imprimir(f"Olá! Meu nome é {proprio.nome} e sou um {proprio.raca}")

# Criando objetos (instâncias) da classe
rex = Cachorro("Rex", "Labrador")
bob = Cachorro("Bob", "Poodle")

# Usando os métodos
rex.latir()  # Rex: Au au!
bob.apresentar()  # Olá! Meu nome é Bob e sou um Poodle
```

Explicação: - `classe` define o molde - `__init__` é o construtor - inicializa o objeto - `proprio` se refere ao próprio objeto (como "eu mesmo") - `proprio.nome` é um atributo do objeto

Classe Conta Bancária:

```
classe ContaBancaria:
    definir __init__(proprio, titular, saldo_inicial=0):
        proprio.titular = titular
        proprio.saldo = saldo_inicial

    definir depositar(proprio, valor):
        proprio.saldo = proprio.saldo + valor
```

```
        imprimir(f"Depósito de R$ {valor:.2f} realizado!")
        proprio.mostrar_saldo()

    definir sacar(proprio, valor):
        se valor > proprio.saldo:
            imprimir("Saldo insuficiente!")
        senao:
            proprio.saldo = proprio.saldo - valor
            imprimir(f"Saque de R$ {valor:.2f} realizado!")
            proprio.mostrar_saldo()

    definir mostrar_saldo(proprio):
        imprimir(f"Saldo atual de {proprio.titular}: R$ {proprio.saldo:.2f}")

# Criando contas
conta_joao = ContaBancaria("João", 1000)
conta_maria = ContaBancaria("Maria", 500)

# Operações
conta_joao.mostrar_saldo()
conta_joao.depositar(500)
conta_joao.sacar(200)

conta_maria.mostrar_saldo()
conta_maria.sacar(600)  # Vai dar erro de saldo insuficiente
```

Classe Retângulo - Exemplo Matemático:

```
classe Retangulo:
    definir __init__(proprio, largura, altura):
        proprio.largura = largura
        proprio.altura = altura

    definir calcular_area(proprio):
        area = proprio.largura * proprio.altura
        retornar area

    definir calcular_perimetro(proprio):
        perimetro = 2 * (proprio.largura + proprio.altura)
        retornar perimetro

    definir mostrar_info(proprio):
        imprimir(f"=== Retângulo ===")
        imprimir(f"Largura: {proprio.largura}")
        imprimir(f"Altura: {proprio.altura}")
        imprimir(f"Área: {proprio.calcular_area()}")
        imprimir(f"Perímetro: {proprio.calcular_perimetro()}")

# Criando retângulos
```

```
ret1 = Retangulo(5, 3)
ret2 = Retangulo(10, 7)

ret1.mostrar_info()
ret2.mostrar_info()
```

Classe Aluno - Sistema Escolar:

```
classe Aluno:
    definir __init__(proprio, nome, matricula):
        proprio.nome = nome
        proprio.matricula = matricula
        proprio.notas = []

    definir adicionar_nota(proprio, nota):
        proprio.notas.append(nota)
        imprimir(f"Nota {nota} adicionada para {proprio.nome}")

    definir calcular_media(proprio):
        se tamanho(proprio.notas) == 0:
            retornar 0
        soma = sum(proprio.notas)
        media = soma / tamanho(proprio.notas)
        retornar media

    definir situacao(proprio):
        media = proprio.calcular_media()
        imprimir(f"\n=== Boletim de {proprio.nome} ===")
        imprimir(f"Matrícula: {proprio.matricula}")
        imprimir(f"Notas: {proprio.notas}")
        imprimir(f"Média: {media:.2f}")

        se media >= 7:
            imprimir("Status: APROVADO ✓")
        senaose media >= 5:
            imprimir("Status: RECUPERAÇÃO ⚠")
        senao:
            imprimir("Status: REPROVADO ✗")

# Usando a classe
aluno1 = Aluno("Carlos Silva", "2024001")
aluno1.adicionar_nota(8.5)
aluno1.adicionar_nota(7.0)
aluno1.adicionar_nota(9.0)
aluno1.situacao()

aluno2 = Aluno("Ana Santos", "2024002")
aluno2.adicionar_nota(6.0)
aluno2.adicionar_nota(5.5)
```

```
aluno2.adicionar_nota(4.0)
aluno2.situacao()
```

Projetos Práticos

 Arquivos de exemplo: exercicios/20-projeto-lista-tarefas.pybr, exercicios/21-projeto-quiz.pybr, exercicios/22-projeto-conversor-temperatura.pybr e exercicios/23-funcoes-avancadas.pybr

Projeto 1: Lista de Tarefas

```
classe GerenciadorTarefas:
    definir __init__(proprio):
        proprio.tarefas = []

    definir adicionar(proprio, tarefa):
        proprio.tarefas.append(tarefa)
        imprimir(f"✓ Tarefa '{tarefa}' adicionada!")

    definir listar(proprio):
        se tamanho(proprio.tarefas) == 0:
            imprimir("Nenhuma tarefa na lista!")
            retornar

        imprimir("\n=== MINHAS TAREFAS ===")
        para i em intervalo(tamanho(proprio.tarefas)):
            imprimir(f"{i + 1}. {proprio.tarefas[i]}")

    definir remover(proprio, numero):
        se numero > 0 e numero <= tamanho(proprio.tarefas):
            tarefa_removida = proprio.tarefas.pop(numero - 1)
            imprimir(f"✓ Tarefa '{tarefa_removida}' removida!")
        senao:
            imprimir("Número inválido!")

# Programa principal
gerenciador = GerenciadorTarefas()

enquanto Verdadeiro:
    imprimir("\n=== MENU ===")
    imprimir("1. Adicionar tarefa")
    imprimir("2. Listar tarefas")
    imprimir("3. Remover tarefa")
    imprimir("4. Sair")

    opcao = entrada("\nEscolha uma opção: ")

    se opcao == "1":
```

```
        tarefa = entrada("Digite a tarefa: ")
        gerenciador.adicionar(tarefa)
    senaose opcao == "2":
        gerenciador.listar()
    senaose opcao == "3":
        gerenciador.listar()
        numero = inteiro(entrada("Digite o número da tarefa para remover: "))
        gerenciador.remover(numero)
    senaose opcao == "4":
        imprimir("Até logo!")
        quebre
    senao:
        imprimir("Opção inválida!")
```

Projeto 2: Jogo de Perguntas e Respostas

```
classe JogoQuiz:
    definir __init__(proprio):
        proprio.pontos = 0
        proprio.perguntas = [
            {
                "pergunta": "Qual é a capital do Brasil?",
                "opcoes": ["1. Rio de Janeiro", "2. São Paulo", "3. Brasília", "4. Salvador"
                "resposta": "3"
            },
            {
                "pergunta": "Quanto é 5 + 7?",
                "opcoes": ["1. 10", "2. 11", "3. 12", "4. 13"],
                "resposta": "3"
            },
            {
                "pergunta": "Qual é a cor do céu?",
                "opcoes": ["1. Verde", "2. Azul", "3. Vermelho", "4. Amarelo"],
                "resposta": "2"
            }
        ]

    definir jogar(proprio):
        imprimir("=== QUIZ - TESTE SEUS CONHECIMENTOS ===\n")

        para i em intervalo(tamanho(proprio.perguntas)):
            pergunta = proprio.perguntas[i]
            imprimir(f"\nPergunta {i + 1}: {pergunta['pergunta']}")

            para opcao em pergunta['opcoes']:
                imprimir(opcao)

            resposta = entrada("\nSua resposta: ")
```

```
            se resposta == pergunta['resposta']:
                imprimir("✓ Correto!")
                proprio.pontos = proprio.pontos + 1
            senao:
                imprimir("✗ Errado!")

        proprio.mostrar_resultado()

    definir mostrar_resultado(proprio):
        total = tamanho(proprio.perguntas)
        imprimir(f"\n=== RESULTADO FINAL ===")
        imprimir(f"Você acertou {proprio.pontos} de {total} perguntas")

        percentual = (proprio.pontos / total) * 100

        se percentual == 100:
            imprimir("□ Perfeito! Você é um gênio!")
        senaose percentual >= 70:
            imprimir("😊 Muito bem! Bom desempenho!")
        senaose percentual >= 50:
            imprimir("😐 Razoável. Estude mais!")
        senao:
            imprimir("😞 Precisa estudar mais!")

# Iniciando o jogo
jogo = JogoQuiz()
jogo.jogar()
```

Projeto 3: Conversor de Temperaturas

```
classe ConversorTemperatura:
    definir celsius_para_fahrenheit(proprio, celsius):
        fahrenheit = (celsius * 9/5) + 32
        retornar fahrenheit

    definir fahrenheit_para_celsius(proprio, fahrenheit):
        celsius = (fahrenheit - 32) * 5/9
        retornar celsius

    definir celsius_para_kelvin(proprio, celsius):
        kelvin = celsius + 273.15
        retornar kelvin

    definir kelvin_para_celsius(proprio, kelvin):
        celsius = kelvin - 273.15
        retornar celsius

    definir menu(proprio):
        enquanto Verdadeiro:
```

```
            imprimir("\n=== CONVERSOR DE TEMPERATURA ===")
            imprimir("1. Celsius → Fahrenheit")
            imprimir("2. Fahrenheit → Celsius")
            imprimir("3. Celsius → Kelvin")
            imprimir("4. Kelvin → Celsius")
            imprimir("5. Sair")

            opcao = entrada("\nEscolha: ")

            se opcao == "5":
                imprimir("Até logo!")
                quebre

            se opcao em ["1", "2", "3", "4"]:
                valor = flutuante(entrada("Digite a temperatura: "))

                se opcao == "1":
                    resultado = proprio.celsius_para_fahrenheit(valor)
                    imprimir(f"{valor}°C = {resultado:.2f}°F")
                senaose opcao == "2":
                    resultado = proprio.fahrenheit_para_celsius(valor)
                    imprimir(f"{valor}°F = {resultado:.2f}°C")
                senaose opcao == "3":
                    resultado = proprio.celsius_para_kelvin(valor)
                    imprimir(f"{valor}°C = {resultado:.2f}K")
                senaose opcao == "4":
                    resultado = proprio.kelvin_para_celsius(valor)
                    imprimir(f"{valor}K = {resultado:.2f}°C")
            senao:
                imprimir("Opção inválida!")

# Executando
conversor = ConversorTemperatura()
conversor.menu()
```

Dicas Finais para Iniciantes

1. Pratique Todos os Dias

- Faça pequenos programas
- Modifique os exemplos
- Crie suas próprias versões

2. Cometa Erros!

- Erros são parte do aprendizado
- Leia as mensagens de erro
- Tente entender o que deu errado

3. Comece Simples

- Não tente fazer tudo de uma vez
- Divida problemas grandes em partes pequenas
- Teste cada parte separadamente

4. Comente Seu Código

```
# Isso é um comentário - explica o que o código faz
# O computador ignora comentários

# Calculando a média
nota1 = 8.0
nota2 = 7.5
media = (nota1 + nota2) / 2  # Soma e divide por 2
```

5. Use Nomes Descritivos

```
# □ Ruim
x = 10
y = 20
z = x + y

# ✓ Bom
idade_joao = 10
idade_maria = 20
soma_idades = idade_joao + idade_maria
```

6. Teste Frequentemente

- Não escreva muito código sem testar
- Teste pequenas partes por vez
- Use `imprimir()` para verificar valores

Próximos Passos

Agora que você aprendeu os fundamentos, você pode:

1. **Explorar mais recursos do Python/PyBR:**
   - Listas e dicionários mais complexos
   - Arquivos (ler e escrever)
   - Bibliotecas externas
2. **Criar projetos maiores:**
   - Sistema de cadastro
   - Jogo de texto
   - Agenda de contatos
3. **Aprender conceitos avançados:**
   - Herança de classes
   - Tratamento de exceções

- Programação funcional

Recursos Adicionais

- **Pratique no REPL do PyBR:**

  ```
  python pybr.py
  ```

- **Execute seus programas:**

  ```
  python pybr.py meu_programa.pybr
  ```

- **Use a extensão VS Code** para ter syntax highlighting e facilitar a escrita de código

---

Lista Completa de Arquivos de Exemplo

Todos os exemplos práticos estão disponíveis na pasta `exercicios/`. Execute qualquer um deles com:

```
python pybr.py exercicios/[nome-do-arquivo].pybr
```

Fundamentos (01-05)

- `01-ola-mundo.pybr` - Primeiro programa
- `02-variaveis.pybr` - Trabalhando com variáveis
- `03-calculos.pybr` - Operações matemáticas
- `04-entrada-saida.pybr` - Interação com usuário
- `05-calculadora-imc.pybr` - Calculadora de IMC

Controle de Fluxo (06-12)

- `06-condicionais.pybr` - Estruturas condicionais
- `07-sistema-login.pybr` - Sistema de login
- `08-lacos-para.pybr` - Laço PARA (for)
- `09-tabuada.pybr` - Gerador de tabuada
- `10-enquanto.pybr` - Laço ENQUANTO (while)
- `11-jogo-adivinhacao.pybr` - Jogo interativo
- `12-listas.pybr` - Trabalhando com listas

Funções (13-15)

- `13-funcoes-simples.pybr` - Funções básicas
- `14-funcoes-retorno.pybr` - Funções com retorno
- `15-calculadora-funcoes.pybr` - Calculadora completa

Classes (16-19)

- `16-classe-cachorro.pybr` - Primeira classe
- `17-classe-conta-bancaria.pybr` - Sistema bancário
- `18-classe-retangulo.pybr` - Cálculos geométricos

- `19-classe-aluno.pybr` - Sistema escolar

Projetos (20-23)

- `20-projeto-lista-tarefas.pybr` - Gerenciador de tarefas
- `21-projeto-quiz.pybr` - Jogo de perguntas
- `22-projeto-conversor-temperatura.pybr` - Conversor de temperaturas
- `23-funcoes-avancadas.pybr` - Lambda, filtrar, mapear

 Dica: Comece pelos primeiros arquivos e vá progredindo. Cada arquivo é independente e pode ser executado separadamente!

Exercícios Propostos

Nível Iniciante:

1. Faça um programa que pergunte seu nome e idade, e diga quantos anos você terá em 2050
2. Crie uma calculadora de gorjeta (10%, 15% ou 20%)
3. Faça um programa que mostre os números pares de 1 a 20

Nível Intermediário:

4. Crie um conversor de moedas (Real, Dólar, Euro)
5. Faça um programa que calcule o fatorial de um número
6. Crie uma função que verifique se um número é primo

Nível Avançado:

7. Crie uma classe `Livro` para uma biblioteca
8. Faça um jogo de adivinhação com níveis de dificuldade
9. Crie um sistema de cadastro de produtos com preços

Parabéns!

Você completou o guia de programação com PyBR! Continue praticando e criando seus próprios projetos. A programação é uma habilidade que melhora com a prática.

Lembre-se: Todo programador foi iniciante um dia. O importante é não desistir e continuar aprendendo!

Bons códigos!