Introdução ao Python

A linguagem **Python** foi criada no final da década de 1980 e lançada oficialmente em **1991** por **Guido van Rossum**, um programador holandês. Ele trabalhava no CWI (Centrum Wiskunde & Informatica) na Holanda e queria desenvolver uma linguagem de programação mais acessível, fácil de ler e escrever, mas ainda poderosa o suficiente para resolver problemas reais.

Origem e Motivação

Guido van Rossum era um fã da linguagem **ABC**, que era fácil de aprender e tinha algumas ideias interessantes, como manipulação de strings e listas. No entanto, a ABC tinha algumas limitações, como a falta de extensibilidade. Ele decidiu então criar uma nova linguagem que resolvesse esses problemas e fosse mais prática para programadores profissionais.

O projeto começou no **Natal de 1989**, quando Guido queria um projeto divertido para ocupar seu tempo durante as férias. Ele se inspirou em C, Modula-3 e outras linguagens, mas manteve um foco na legibilidade e simplicidade.

Por que o nome "Python"?

O nome **Python** não tem nada a ver com cobras! 🐍 Na verdade, Guido van Rossum era fã do grupo de comédia britânico **Monty Python's Flying Circus**. Ele queria um nome curto, único e um pouco misterioso, então escolheu "Python".

Primeiros Anos (1991 - 2000)

- **1991**: Guido lançou a **versão 0.9.0** do Python no grupo de discussão Usenet alt.sources.
  - Já incluía **classes, exceções, módulos e tipos de dados como listas e dicionários**.
- **1994**: Criada a **Python Software Foundation (PSF)**, que passou a cuidar do desenvolvimento da linguagem.
- **2000**: Lançamento do **Python 2.0**, trazendo melhorias como **coleta de lixo automática** e **listas por compreensão (list comprehensions)**.

  ⚠ **Problema do Python 2**: Ele não era totalmente compatível com o Python 3, o que gerou dificuldades para os desenvolvedores por muitos anos.

Evolução com Python 3 (2008 - Presente)

- **2008**: Lançamento do **Python 3.0**, uma reformulação para tornar a linguagem mais consistente e moderna.
  - Melhorias na manipulação de strings (Unicode por padrão).
  - Alterações na sintaxe (exemplo: print virou função → print("Olá, mundo!")).
- **2020**: O suporte ao **Python 2 foi oficialmente encerrado**.
- **Atualmente**: Python é uma das linguagens mais populares do mundo, sendo amplamente usado em:
  - **Ciência de Dados e Machine Learning** (com pandas, NumPy, scikit-learn, TensorFlow).
  - **Desenvolvimento Web** (com frameworks como Django e Flask).
  - **Automação e Scripts**.

- **Cibersegurança e Hacking Ético**.
- **Jogos e Aplicações Desktop**.

Python Hoje e no Futuro

- Python se tornou a **linguagem mais popular** segundo rankings como **TIOBE e Stack Overflow Developer Survey**.
- Grandes empresas como **Google, Facebook, NASA e Netflix** usam Python intensivamente.
- A tendência é que o Python continue crescendo, especialmente em **inteligência artificial, automação e desenvolvimento web**.

Python surgiu como um projeto pessoal de **Guido van Rossum**, mas hoje é uma das linguagens mais usadas no mundo devido à sua simplicidade e versatilidade.

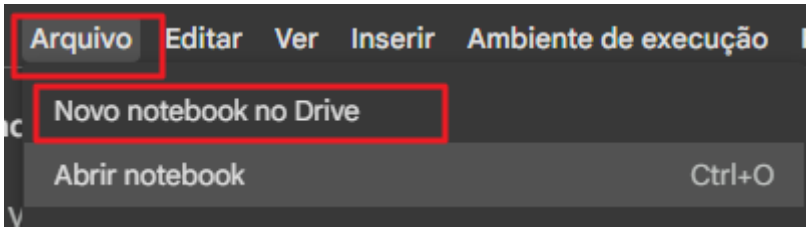
Python é uma linguagem de programação de alto nível, fácil de aprender e amplamente utilizada em diversos campos, como desenvolvimento web, análise de dados, inteligência artificial, automação, entre outros.

Sua sintaxe simples e a vasta biblioteca padrão tornam-na uma excelente escolha tanto para iniciantes quanto para programadores experientes.

A seguir, abordaremos uma série de exemplos didáticos para que você compreenda os conceitos fundamentais da linguagem Python e aprenda a programar de maneira prática e eficiente.

Vamos praticar

Acesse - https://colab.research.google.com/#scrollTo=Wf5KrEb6vrkR

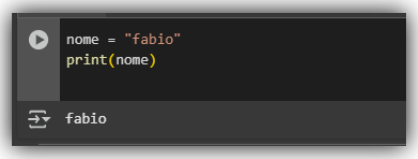


Passo a Passo

1. Imprimindo uma mensagem simples

Python

```
nome = "fabio"
print(nome)
```



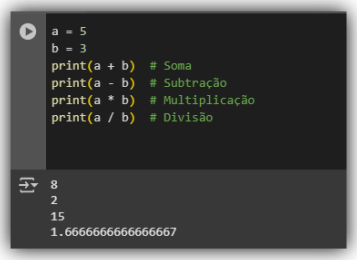
Este código mostra como usar a função print() para exibir informações no console. O valor da variável nome será impresso.

---

2. Operações matemáticas básicas

Python

```
a = 5
b = 3
print(a + b)  # Soma
print(a - b)  # Subtração
print(a * b)  # Multiplicação
print(a / b)  # Divisão
```



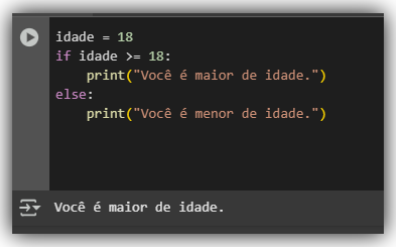
Aqui, você aprende a realizar operações matemáticas básicas: soma, subtração, multiplicação e divisão.

---

3. Usando estruturas condicionais (if-else)

Python

```
idade = 18
if idade >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")
```



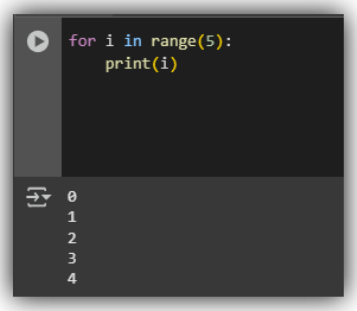
As estruturas condicionais (if-else) permitem que você execute diferentes blocos de código dependendo de uma condição.

---

4. Loops (for loop)

Python

```
for i in range(5):
    print(i)
```



O for loop é usado para iterar sobre uma sequência (como uma lista, tupla ou intervalo de números).

---

5. Loops (while loop)

Python

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

O while loop executa o código enquanto uma condição for verdadeira. No exemplo, ele continua até que o contador chegue a 5.

---

6. Definindo funções

Python

```
def saudacao(nome):
    print(f"Olá, {nome}!")

saudacao("Fabio")
```

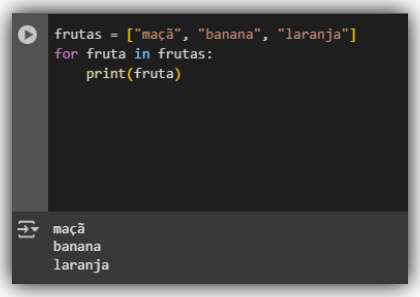
Funções em Python permitem que você agrupe código reutilizável. A função saudacao recebe um parâmetro e exibe uma mensagem personalizada.

---

7. Listas

Python

```
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)
```



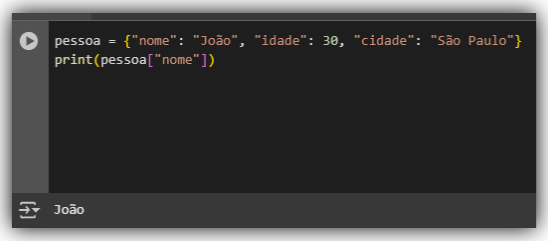
Listas são coleções ordenadas e mutáveis.
Você pode usar um loop para acessar seus elementos.

---

8. Dicionários

Python

```
pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"])
```

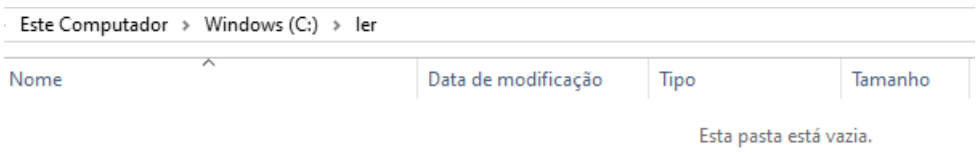


Dicionários são coleções não ordenadas de pares chave-valor.

Eles são úteis quando você precisa associar uma chave a um valor.

---

9. Manipulação de arquivos



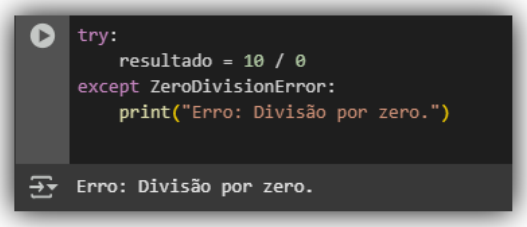
```
import os
# Obtendo o caminho para a pasta Downloads

downloads_path = os.path.join(os.path.expanduser("~"), "Downloads", "arquivo.txt")

# Abrindo e escrevendo no arquivo
with open(downloads_path, "w") as arquivo:arquivo.write("Olá, arquivo!")
```

A manipulação de arquivos permite que você leia e escreva em arquivos de texto. No exemplo, estamos criando e escrevendo em um arquivo chamado arquivo.txt.

---

10. Manipulando exceções

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: Divisão por zero.")
```



O tratamento de exceções ajuda a lidar com erros durante a execução do código, evitando que o programa quebre

---

11. Verificando tipo de variável

variavel = 10
print(type(variavel)) # <class 'int'>

```
variavel = 10
print(type(variavel))  # <class 'int'>
```

```
<class 'int'>
```

A função type() retorna o tipo de dado de uma variável.
No caso, int significa inteiro.

---

12. Acessando elementos de uma lista

lista = [1, 2, 3, 4]
print(lista[2]) # Acessa o terceiro elemento, que é 3

```
lista = [1, 2, 3, 4]
print(lista[2])  # Acessa o terceiro elemento, que é 3
```

```
3
```

As listas em Python são indexadas, e você pode acessar seus elementos usando um índice

---

13. Modificando elementos de uma lista

lista[1] = 10
print(lista) # [1, 10, 3, 4]

```
lista[1] = 10
print(lista)  # [1, 10, 3, 4]
```

```
[1, 10, 3, 4]
```

Você pode modificar os elementos de uma lista diretamente, usando o índice do elemento.

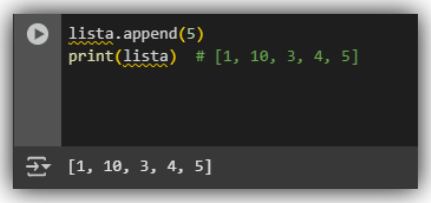
---

14. Adicionando elementos a uma lista

Python

lista.append(5)

print(lista) # [1, 10, 3, 4, 5]

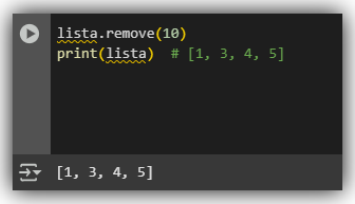


O método append() adiciona um novo elemento ao final de uma lista.

---

15. Removendo elementos de uma lista

lista.remove(10)
print(lista) # [1, 3, 4, 5]

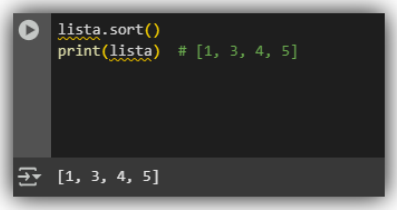


O método remove() remove a primeira ocorrência de um valor específico em uma lista.

---

16. Ordenando uma lista

lista.sort()
print(lista) # [1, 3, 4, 5]



O método sort() ordena os elementos de uma lista em ordem crescente.

---

17. Desempacotamento de listas

a, b, c = [1, 2, 3]
print(a, b, c) # 1 2 3

```python
a, b, c = [1, 2, 3]
print(a, b, c)  # 1 2 3
```

```
1 2 3
```

O desempacotamento de listas permite atribuir os elementos de uma lista a variáveis.

---

18. Concatenando listas

lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa) # [1, 2, 3, 4, 5, 6]

```python
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)  # [1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

A concatenação de listas é feita usando o operador +.

---

19. Tuplas

tupla = (1, 2, 3)
print(tupla[0]) # 1

```python
tupla = (1, 2, 3)
print(tupla[0])  # 1
```

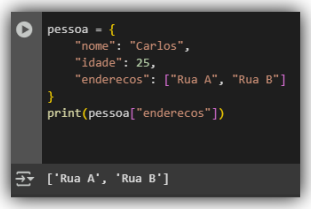
```
1
```

As tuplas são semelhantes às listas, mas são imutáveis. Ou seja, você não pode modificar seus elementos depois de criá-los.

---

20. Dicionários com listas

```
pessoa = {
    "nome": "Carlos",
    "idade": 25,
    "enderecos": ["Rua A", "Rua B"]
}
print(pessoa["enderecos"])
```



Os dicionários podem conter listas como valores. Você pode acessar as listas de maneira similar.

Neste passo a passo, exploramos os conceitos fundamentais de Python, incluindo operações matemáticas, estruturas condicionais, loops, funções, listas, dicionários e manipulação de arquivos.

Com esses conceitos, você já pode começar a escrever scripts mais complexos e resolver problemas de programação de forma mais eficiente.

Estes exemplos podem ser usados como base para entender e praticar a linguagem Python.

Experimente modificar os exemplos e criar suas próprias variações para se aprofundar no aprendizado!

Codigos para Praticar

```
nome = "fabio";
print(nome);

# Exemplo 1: Imprimir uma mensagem simples
print("Olá, mundo!")

# Exemplo 2: Operações matemáticas básicas
a = 5
b = 3
print(a + b)  # Soma
print(a - b)  # Subtração
print(a * b)  # Multiplicação
print(a / b)  # Divisão

# Exemplo 3: Usando estruturas condicionais (if-else)
idade = 18
```

```
if idade >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")

# Exemplo 4: Loops (for loop)
for i in range(5):
    print(i)

# Exemplo 5: Loops (while loop)
contador = 0
while contador < 5:
    print(contador)
    contador += 1

# Exemplo 6: Definindo funções
def saudacao(nome):
    print(f"Olá, {nome}!")

saudacao("Maria")

# Exemplo 7: Listas
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)

# Exemplo 8: Dicionários
pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"])


# Exemplo 9: Manipulação de arquivos
import os
# Obtendo o caminho para a pasta Downloads
downloads_path = os.path.join(os.path.expanduser("~"), "Downloads", "arquivo.txt")
# Abrindo e escrevendo no arquivo
with open(downloads_path, "w") as arquivo:arquivo.write("Olá, arquivo!")


# Exemplo 10: Manipulando exceções
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: Divisão por zero.")


# Exemplo 11: Verificando tipo de variável
variavel = 10
print(type(variavel))  # <class 'int'>

# Exemplo 12: Acessando elementos de uma lista
lista = [1, 2, 3, 4]
print(lista[2])  # Acessa o terceiro elemento, que é 3

# Exemplo 13: Modificando elementos de uma lista
lista[1] = 10
```

```
print(lista)  # [1, 10, 3, 4]

# Exemplo 14: Adicionando elementos a uma lista
lista.append(5)
print(lista)  # [1, 10, 3, 4, 5]

# Exemplo 15: Removendo elementos de uma lista
lista.remove(10)
print(lista)  # [1, 3, 4, 5]

# Exemplo 16: Ordenando uma lista
lista.sort()
print(lista)  # [1, 3, 4, 5]

# Exemplo 17: Desempacotamento de listas
a, b, c = [1, 2, 3]
print(a, b, c)  # 1 2 3

# Exemplo 18: Concatenando listas
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)  # [1, 2, 3, 4, 5, 6]

# Exemplo 19: Tuplas
tupla = (1, 2, 3)
print(tupla[0])  # 1

# Exemplo 20: Dicionários com listas
pessoa = {
    "nome": "Carlos",
    "idade": 25,
    "enderecos": ["Rua A", "Rua B"]
}
print(pessoa["enderecos"])

# Exemplo 21: Definindo funções com parâmetros
def soma(a, b):
    return a + b

resultado = soma(5, 10)
print(resultado)  # 15

# Exemplo 22: Função com valor de retorno
def quadrado(x):
    return x ** 2

print(quadrado(4))  # 16

# Exemplo 23: Funções com argumentos padrões
def saudacao(nome="Visitante"):
    print(f"Olá, {nome}!")

saudacao("Ana")
saudacao()  # Olá, Visitante!
```

```python
# Exemplo 24: Função recursiva
def fatorial(n):
    if n == 1:
        return 1
    else:
        return n * fatorial(n - 1)

print(fatorial(5))  # 120

# Exemplo 25: Lambdas (funções anônimas)
soma = lambda a, b: a + b
print(soma(3, 4))  # 7

# Exemplo 26: List comprehension
quadrados = [x ** 2 for x in range(5)]
print(quadrados)  # [0, 1, 4, 9, 16]

# Exemplo 27: Funções de map
numeros = [1, 2, 3, 4]
quadrados = list(map(lambda x: x ** 2, numeros))
print(quadrados)  # [1, 4, 9, 16]

# Exemplo 28: Função de filtro
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(pares)  # [2, 4]

# Exemplo 29: Função reduce
from functools import reduce
soma_total = reduce(lambda x, y: x + y, numeros)
print(soma_total)  # 10

# Exemplo 30: Compreensão de dicionários
dicionario = {x: x ** 2 for x in range(5)}
print(dicionario)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Exemplo 31: Acessando caracteres de uma string
texto = "Python"
print(texto[0])  # 'P'

# Exemplo 32: Fatiamento de strings
print(texto[1:4])  # 'yth'

# Exemplo 33: Concatenando strings
nome = "Ana"
sobrenome = "Silva"
nome_completo = nome + " " + sobrenome
print(nome_completo)  # 'Ana Silva'

# Exemplo 34: Métodos de string
frase = "olá mundo"
print(frase.upper())  # 'OLÁ MUNDO'

# Exemplo 35: Substituindo palavras em uma string
texto = "Eu gosto de Python"
novo_texto = texto.replace("Python", "Java")
```

```
print(novo_texto)  # 'Eu gosto de Java'

# Exemplo 36: Dividindo uma string
texto = "Python é ótimo"
palavras = texto.split()
print(palavras)  # ['Python', 'é', 'ótimo']

# Exemplo 37: Verificando se uma substring está presente
print("Python" in texto)  # True

# Exemplo 38: Função join para concatenar lista de strings
palavras = ["Python", "é", "legal"]
frase = " ".join(palavras)
print(frase)  # 'Python é legal'

# Exemplo 39: Usando o operador "in" com listas
numeros = [1, 2, 3, 4, 5]
print(3 in numeros)  # True

# Exemplo 40: Manipulando conjuntos
conjunto = {1, 2, 3}
conjunto.add(4)
print(conjunto)  # {1, 2, 3, 4}

# Exemplo 41: Diferença entre listas e conjuntos
lista = [1, 2, 3, 3, 4]
conjunto = set(lista)
print(conjunto)  # {1, 2, 3, 4} - Conjunto remove duplicatas

# Exemplo 42: Criando e manipulando arquivos
with open("meuarquivo.txt", "w") as arquivo:
    arquivo.write("Python é uma linguagem poderosa!")

with open("meuarquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)  # 'Python é uma linguagem poderosa!'

# Exemplo 43: Manipulando diretórios
import os
os.mkdir("meu_diretorio")
print("Diretório criado!")

# Exemplo 44: Verificando se um arquivo ou diretório existe
existe = os.path.exists("meuarquivo.txt")
print(existe)  # True

# Exemplo 45: Listando arquivos em um diretório
arquivos = os.listdir(".")
print(arquivos)  # Listagem de arquivos no diretório atual

# Exemplo 46: Utilizando a biblioteca math
import math
print(math.sqrt(16))  # 4.0
```

```
# Exemplo 47: Gerando números aleatórios
import random
print(random.randint(1, 10))  # Número aleatório entre 1 e 10

# Exemplo 48: Trabalhando com data e hora
import datetime
hoje = datetime.datetime.now()
print(hoje)  # Exibe a data e hora atual

# Exemplo 49: Tratamento de erros com try-except
try:
    numero = int(input("Digite um número: "))
except ValueError:
    print("Erro: Entrada inválida!")

# Exemplo 50: Função de ajuda (help)
help(print)  # Exibe a documentação da função print
```

Explicando

1. Imprimir uma mensagem simples

A função print() é usada para exibir mensagens ou valores na tela. Este exemplo mostra como imprimir uma simples saudação.

---

2. Operações matemáticas básicas

Neste exemplo, mostramos como realizar operações matemáticas como soma, subtração, multiplicação e divisão entre dois números. O resultado de cada operação é impresso na tela.

---

3. Usando estruturas condicionais (if-else)

As estruturas condicionais (if-else) permitem executar diferentes blocos de código com base em uma condição. Este exemplo verifica se uma pessoa é maior ou menor de idade.

---

4. Loops (for loop)

O loop for é utilizado para repetir uma ação um número específico de vezes ou para iterar sobre uma sequência de valores. Neste caso, o loop imprime números de 0 a 4.

---

5. Loops (while loop)

O loop while repete um bloco de código enquanto uma condição for verdadeira. Este exemplo utiliza um contador para imprimir números de 0 a 4.

---

6. Definindo funções

Uma função é um bloco de código que pode ser reutilizado. Este exemplo define uma função saudacao() que recebe um nome como parâmetro e imprime uma saudação personalizada.

---

7. Listas

Listas são coleções de elementos ordenados. Este exemplo percorre uma lista de frutas e imprime cada uma delas.

---

8. Dicionários

Dicionários são coleções de pares chave-valor. Aqui, é mostrado como acessar um valor dentro de um dicionário, usando uma chave específica.

---

9. Manipulação de arquivos

Este exemplo mostra como criar e escrever em um arquivo de texto dentro da pasta **Downloads** do usuário. Ele usa o método open() para abrir ou criar um arquivo e escrever nele.

---

10. Manipulando exceções

O tratamento de exceções permite capturar erros durante a execução do código, evitando que o programa pare de funcionar. Este exemplo trata um erro de divisão por zero.

---

11. Verificando tipo de variável

O Python permite verificar o tipo de uma variável com a função type(). Este exemplo imprime o tipo de uma variável inteira.

---

12. Acessando elementos de uma lista

As listas são indexadas, o que significa que você pode acessar seus elementos usando índices. Este exemplo mostra como acessar o terceiro elemento de uma lista.

---

13. Modificando elementos de uma lista

Você pode alterar os valores dentro de uma lista acessando um índice específico. Este exemplo modifica o segundo item de uma lista.

---

14. Adicionando elementos a uma lista

Você pode adicionar elementos ao final de uma lista usando o método append(). Aqui, é mostrado como adicionar um número ao final da lista.

---

15. Removendo elementos de uma lista

A função remove() permite remover o primeiro elemento que corresponde ao valor fornecido. Este exemplo remove um item específico de uma lista.

---

16. Ordenando uma lista

Listas podem ser ordenadas em ordem crescente com o método sort(). Este exemplo mostra como ordenar os números de uma lista.

---

17. Desempacotamento de listas

O desempacotamento de listas é uma maneira de atribuir múltiplos valores de uma lista a variáveis separadas. Este exemplo mostra como atribuir três valores de uma lista a três variáveis.

---

18. Concatenando listas

Listas podem ser combinadas usando o operador +, o que cria uma nova lista contendo os elementos de ambas. Este exemplo concatena duas listas.

---

19. Tuplas

As tuplas são coleções imutáveis. Este exemplo mostra como acessar elementos de uma tupla. Diferente das listas, as tuplas não podem ser alteradas após a criação.

---

20. Dicionários com listas

Os dicionários podem armazenar listas como valores. Este exemplo mostra como acessar uma lista dentro de um dicionário.

---

21. Definindo funções com parâmetros

Funções podem receber parâmetros para operar de maneira dinâmica. Este exemplo define uma função que recebe dois números e retorna sua soma.

---

22. Função com valor de retorno

As funções podem retornar valores, que podem ser usados em outras partes do código. Este exemplo define uma função que retorna o quadrado de um número.

---

23. Funções com argumentos padrões

Funções podem ter valores padrão para seus parâmetros. Este exemplo mostra como a função pode ser chamada com ou sem um argumento, usando um valor padrão se o argumento não for fornecido.

---

24. Função recursiva

Funções recursivas são aquelas que chamam a si mesmas. Este exemplo define uma função que calcula o fatorial de um número, utilizando recursão.

---

25. Lambdas (funções anônimas)

Funções lambda são funções pequenas e anônimas que podem ser definidas em uma única linha. Este exemplo mostra uma função lambda que soma dois números.

---

26. List comprehension

A list comprehension é uma maneira compacta de criar listas. Este exemplo cria uma lista dos quadrados dos números de 0 a 4.

---

27. Funções de map

A função map() aplica uma função a todos os itens de uma sequência, retornando uma nova sequência com os resultados. Este exemplo calcula o quadrado de todos os números de uma lista.

---

28. Função de filtro

A função filter() filtra elementos de uma sequência com base em uma condição fornecida. Este exemplo seleciona os números pares de uma lista.

29. Função reduce

A função reduce() aplica uma função acumuladora a uma sequência, reduzindo-a a um único valor. Este exemplo soma todos os números de uma lista.

---

30. Compreensão de dicionários

Assim como a list comprehension, você também pode criar dicionários de forma concisa. Este exemplo cria um dicionário com números e seus quadrados.

---

31. Acessando caracteres de uma string

As strings são sequências de caracteres e você pode acessar um caractere específico usando índices. Este exemplo acessa o primeiro caractere de uma string.

---

32. Fatiamento de strings

O fatiamento de strings permite que você acesse um intervalo de caracteres. Este exemplo mostra como obter uma parte de uma string.

---

33. Concatenando strings

Strings podem ser concatenadas (unidas) com o operador +. Este exemplo concatena o nome e o sobrenome para formar um nome completo.

---

34. Métodos de string

As strings possuem muitos métodos úteis, como upper(), que converte todos os caracteres em maiúsculas. Este exemplo converte uma frase para maiúsculas.

---

35. Substituindo palavras em uma string

O método replace() permite substituir uma palavra ou parte de uma string por outra. Este exemplo substitui a palavra "Python" por "Java".

---

36. Dividindo uma string

O método split() divide uma string em uma lista de substrings, usando um separador. Este exemplo divide uma frase em palavras.

37. Verificando se uma substring está presente

Você pode verificar se uma substring está presente em uma string com o operador in. Este exemplo verifica se "Python" está presente em uma string.

---

38. Função join para concatenar lista de strings

O método join() permite concatenar todos os elementos de uma lista de strings, inserindo um separador entre eles. Este exemplo junta uma lista de palavras em uma frase.

---

39. Usando o operador "in" com listas

O operador in também pode ser usado com listas para verificar se um item está presente. Este exemplo verifica se o número 3 está na lista.

---

40. Manipulando conjuntos

Conjuntos são coleções de elementos únicos. O método add() permite adicionar um novo elemento a um conjunto. Este exemplo adiciona o número 4 a um conjunto.

---

41. Diferença entre listas e conjuntos

A principal diferença entre listas e conjuntos é que os conjuntos não permitem elementos duplicados. Este exemplo converte uma lista em um conjunto, removendo duplicatas.

---

42. Criando e manipulando arquivos

Este exemplo mostra como criar e escrever em um arquivo, e depois ler o conteúdo do arquivo. A manipulação de arquivos é útil para armazenar dados.

---

43. Manipulando diretórios

A função os.mkdir() permite criar novos diretórios. Este exemplo cria um diretório chamado meu_diretorio.

---

44. Verificando se um arquivo ou diretório existe

O método os.path.exists() verifica se um arquivo ou diretório existe no caminho especificado. Este exemplo verifica se o arquivo meuarquivo.txt existe.

45. Listando arquivos em um diretório

O método os.listdir() lista todos os arquivos e pastas presentes em um diretório. Este exemplo lista os arquivos no diretório atual.

---

46. Utilizando a biblioteca math

A biblioteca math oferece várias funções matemáticas, como sqrt(), que calcula a raiz quadrada. Este exemplo calcula a raiz quadrada de 16.

---

47. Gerando números aleatórios

A função random.randint() gera números inteiros aleatórios dentro de um intervalo especificado. Este exemplo gera um número aleatório entre 1 e 10.

---

48. Trabalhando com data e hora

A biblioteca datetime permite trabalhar com datas e horas. Este exemplo obtém a data e hora atual e a imprime.

---

49. Tratamento de erros com try-except

O bloco try-except captura e lida com erros durante a execução do código. Este exemplo trata o erro de entrada inválida ao tentar converter uma entrada para um número.

---

50. Função de ajuda (help)

A função help() exibe a documentação de qualquer função, módulo ou objeto. Este exemplo exibe a documentação da função print().

Projeto

Esse projeto representa um **Gerenciador de Tarefas** onde é possível adicionar, listar e remover tarefas. O código está bem comentado para facilitar a explicação na faculdade.

Funcionalidades do projeto:

- Adicionar uma nova tarefa
- Listar todas as tarefas
- Remover uma tarefa pelo índice

Explicação:

1. **Criação da classe GerenciadorDeTarefas**:
   - Possui um atributo tarefas que armazena as tarefas em uma lista.
2. **Método adicionar_tarefa**:
   - Adiciona uma nova tarefa à lista e exibe uma mensagem de confirmação.
3. **Método listar_tarefas**:
   - Percorre a lista e exibe as tarefas de maneira numerada.
   - Se não houver tarefas, exibe uma mensagem informando.
4. **Método remover_tarefa**:
   - Remove uma tarefa da lista com base no índice informado pelo usuário.
   - Garante que o índice fornecido seja válido antes de remover.
5. **Testando a classe**:
   - Criamos uma instância do GerenciadorDeTarefas.
   - Adicionamos três tarefas de exemplo.
   - Listamos as tarefas cadastradas.
   - Removemos uma tarefa específica.
   - Listamos novamente para verificar a remoção.

Codigo

```
class GerenciadorDeTarefas:

    """
    Classe para gerenciar uma lista de tarefas simples.
    Permite adicionar, listar e remover tarefas.
    """

    def __init__(self):
        """Inicializa a lista de tarefas vazia"""
        self.tarefas = []

    def adicionar_tarefa(self, descricao):
        """
        Adiciona uma nova tarefa à lista.

        Parâmetros:
        descricao (str): Descrição da tarefa.
        """
        self.tarefas.append(descricao)
        print(f"Tarefa adicionada: {descricao}")

    def listar_tarefas(self):
        """
        Lista todas as tarefas cadastradas.
        """
        if not self.tarefas:
            print("Nenhuma tarefa cadastrada.")
        else:
            print("\nLista de Tarefas:")
```

```
            for i, tarefa in enumerate(self.tarefas):
                print(f"{i + 1}. {tarefa}")

    def remover_tarefa(self, indice):
        """
        Remove uma tarefa da lista com base no índice informado.

        Parâmetros:
        indice (int): Índice da tarefa a ser removida (começa de 1 para o usuário).
        """
        if 1 <= indice <= len(self.tarefas):
            tarefa_removida = self.tarefas.pop(indice - 1)
            print(f"Tarefa removida: {tarefa_removida}")
        else:
            print("Índice inválido! Tente novamente.")

# Criando uma instância da classe
gerenciador = GerenciadorDeTarefas()

# Adicionando algumas tarefas
gerenciador.adicionar_tarefa("Estudar para a prova de matemática")
gerenciador.adicionar_tarefa("Ler um artigo sobre Inteligência Artificial")
gerenciador.adicionar_tarefa("Fazer exercícios físicos")

# Listando as tarefas
gerenciador.listar_tarefas()

# Removendo uma tarefa
gerenciador.remover_tarefa(2)

# Listando as tarefas novamente
gerenciador.listar_tarefas()
```

Esse código é um ótimo exemplo prático de POO (Programação Orientada a Obje

Introdução ao Java

Origem e Motivação

Java foi criado no início da década de 1990 por uma equipe de engenheiros da Sun Microsystems, liderada por James Gosling. Inicialmente chamado de Oak, o projeto visava o desenvolvimento de dispositivos embarcados e eletrodomésticos inteligentes. No entanto, devido à falta de maturidade desse mercado, o foco foi redirecionado para a Internet, que começava a ganhar popularidade.

A principal motivação para a criação do Java foi desenvolver uma linguagem portável, segura e eficiente, permitindo que programas fossem executados em qualquer sistema operacional sem necessidade de recompilação. Assim surgiu o conceito de "Write Once, Run Anywhere" (Escreva uma vez, execute em qualquer lugar), graças à Java Virtual Machine (JVM), que interpreta o código Java de forma independente do hardware ou sistema operacional.

Primeiras Versões: Java 1.0, 1.1 e 2

A evolução inicial do Java trouxe recursos fundamentais que consolidaram sua popularidade e adoção no mercado de desenvolvimento:

Java 1.0 (1996): Primeira versão oficial, lançada com suporte para aplicações web e applets, além da introdução da JVM, do coletor de lixo (Garbage Collector) e de bibliotecas essenciais para manipulação de gráficos, rede e segurança.

Java 1.1 (1997): Refinou a linguagem com a introdução do JDBC (Java Database Connectivity) para acesso a bancos de dados, RMI (Remote Method Invocation) para chamadas remotas entre objetos e melhorias no modelo de eventos da interface gráfica (AWT).

Java 2 (1998 - 1999): Essa versão marcou uma grande reformulação da plataforma, agora dividida em três edições:

- J2SE (Java 2 Standard Edition) – voltada para aplicações desktop e básicas.
- J2EE (Java 2 Enterprise Edition) – para aplicações corporativas robustas.
- J2ME (Java 2 Micro Edition) – para dispositivos móveis e embarcados.

Além disso, introduziu a Java Collections Framework, aprimorou o gerenciamento de memória e segurança, e solidificou Java como uma plataforma versátil para diversos cenários de desenvolvimento.

Hoje e o Futuro do Java

Atualmente, Java continua sendo uma das linguagens mais utilizadas no mundo, presente em sistemas bancários, aplicações corporativas, desenvolvimento Android, computação em nuvem e até em soluções de Inteligência Artificial e Big Data. A plataforma evoluiu para suportar microservices, com frameworks como Spring Boot, e melhorou sua performance com otimizações na JVM e no Garbage Collector.

Desde a aquisição pela Oracle, o Java passou a adotar um ciclo de lançamentos mais frequente, com versões a cada seis meses. Algumas das inovações recentes incluem:

- Java 17 e Java 21 (LTS) trazendo melhorias de performance e segurança.
- Project Loom, que introduz lightweight threads para melhor concorrência.
- Pattern Matching e Records, para tornar a linguagem mais concisa.

- GraalVM, que permite a compilação antecipada (ahead-of-time), otimizando a execução de código.

Para o futuro, Java continuará a evoluir para acompanhar tendências como computação em nuvem, machine learning e programação reativa, garantindo sua relevância no ecossistema tecnológico global.

Vamos praticar

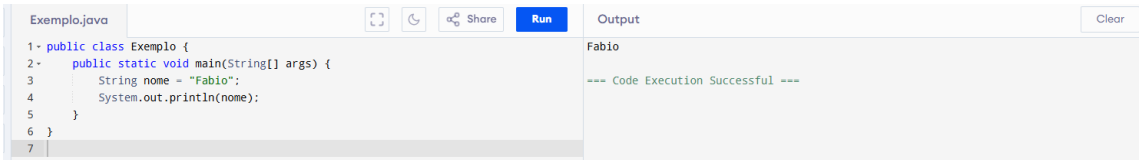
Acesse - https://www.programiz.com/java-programming/online-compiler/

Passo a Passo

1. Imprimindo uma mensagem simples

Em Java, podemos imprimir mensagens no console usando o método **System.out.println()**. Veja um exemplo simples:

```
public class Exemplo {
    public static void main(String[] args) {
        String nome = "Fabio";
        System.out.println(nome);
    }
}
```



- Criamos uma classe chamada **Exemplo**, pois em Java todo código precisa estar dentro de uma classe.
- O método **main()** é o ponto de entrada do programa, onde a execução começa.
- Criamos uma variável **nome** do tipo String e atribuímos o valor "Fabio".
- Usamos **System.out.println(nome);** para imprimir o valor da variável no console.

---

2. Operações matemáticas básicas

Em Java, podemos realizar operações matemáticas básicas da mesma forma que em Python. Veja um exemplo equivalente ao código fornecido:

```
public class OperacoesMatematicas {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;

        System.out.println(a + b); // Soma
        System.out.println(a - b); // Subtração
        System.out.println(a * b); // Multiplicação
        System.out.println((double) a / b); // Divisão (convertendo para double)
    }
```

}

OperacoesMatematicas.java

```java
public class OperacoesMatematicas {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;

        System.out.println(a + b); // Soma
        System.out.println(a - b); // Subtração
        System.out.println(a * b); // Multiplicação
        System.out.println((double) a / b); // Divisão (convertendo para double)
    }
}
```

Output

```
8
2
15
1.6666666666666667

=== Code Execution Successful ===
```

- Criamos uma classe chamada **OperacoesMatematicas**.
- Dentro do método **main()**, declaramos duas variáveis **a** e **b** do tipo int.
- Utilizamos System.out.println() para exibir os resultados das operações:
  - **Soma (+)**
  - **Subtração (-)**
  - **Multiplicação (*)**
  - **Divisão (/)** – Como Java trata divisão entre inteiros como inteiro, usamos (double) a / b para obter um resultado decimal.

---

3. Usando estruturas condicionais (if-else)

Em Java, podemos utilizar a estrutura **if-else** para executar diferentes blocos de código com base em uma condição. Veja o equivalente ao código em Python:

```java
public class VerificaIdade {
    public static void main(String[] args) {
        int idade = 18;

        if (idade >= 18) {
            System.out.println("Você é maior de idade.");
        } else {
            System.out.println("Você é menor de idade.");
        }
    }
}
```

VerificaIdade.java

```java
public class VerificaIdade {
    public static void main(String[] args) {
        int idade = 18;

        if (idade >= 18) {
            System.out.println("Voce eh maior de idade.");
        } else {
            System.out.println("Voce eh menor de idade.");
        }
    }
}
```

Output

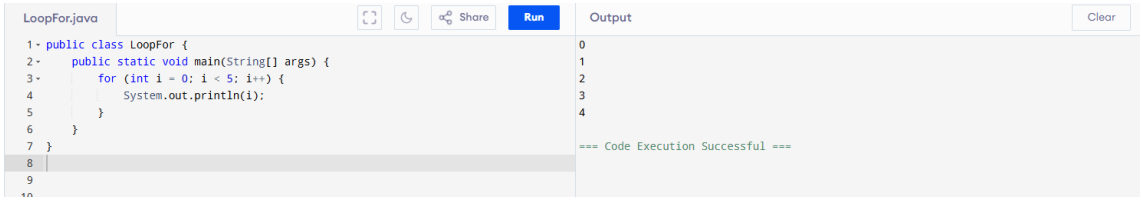
```
Voce eh maior de idade.

=== Code Execution Successful ===
```

- Criamos uma classe chamada **VerificaIdade**.
- No método **main()**, declaramos a variável **idade** do tipo int e atribuímos o valor 18.
- Utilizamos a estrutura **if-else** para verificar se a idade é maior ou igual a 18:
  - Se a condição idade >= 18 for verdadeira, o programa imprime "Você é maior de idade.".
  - Caso contrário, imprime "Você é menor de idade.".

4. Loops (for loop)

Em Java, a estrutura for pode ser usada para iterar sobre um intervalo de números, assim como em Python. Veja o equivalente ao código fornecido:

```
public class LoopFor {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println(i);
        }
    }
}
```



- Criamos uma classe chamada **LoopFor**.
- No método **main()**, utilizamos um **for** para iterar de 0 a 4.
- A estrutura do for em Java segue este formato:

  java
  for (inicialização; condição; incremento) {
      // Bloco de código a ser repetido
  }

  - **int i = 0;** → Inicializa i com o valor 0.
  - **i < 5;** → A condição para continuar o loop é que i seja menor que 5.
  - **i++** → Incrementa i em 1 a cada iteração.
- O método **System.out.println(i);** imprime o valor de i a cada iteração.

---

5. Loops (while loop)

Em Java, o loop while funciona de forma semelhante ao Python, executando um bloco de código enquanto uma condição for verdadeira. Veja o equivalente ao código fornecido:

```
public class LoopWhile {
    public static void main(String[] args) {
        int contador = 0;

        while (contador < 5) {
            System.out.println(contador);
            contador++;
        }
    }
}
```

LoopWhile.java

```java
public class LoopWhile {
    public static void main(String[] args) {
        int contador = 0;

        while (contador < 5) {
            System.out.println(contador);
            contador++;
        }
    }
}
```

Output

```
0
1
2
3
4

=== Code Execution Successful ===
```

- Criamos uma classe chamada **LoopWhile**.
- Dentro do método **main()**, inicializamos a variável **contador** com 0.
- A estrutura do **while** em Java segue este formato:

  java
  CopiarEditar
  while (condição) {
  // Bloco de código a ser repetido
  }

  - **while (contador < 5)** → O código dentro do loop será executado enquanto contador for menor que 5.
  - **System.out.println(contador);** → Imprime o valor do contador.
  - **contador++;** → Incrementa contador em 1 a cada iteração, para evitar um loop infinito.

---

6. Definindo funções

Em Java, funções são chamadas de **métodos** e devem estar dentro de uma classe. O equivalente ao código em Python pode ser escrito da seguinte forma:

```java
public class Saudacao {
    public static void main(String[] args) {
        saudacao("Fabio");
    }

    public static void saudacao(String nome) {
        System.out.println("Olá, " + nome + "!");
    }
}
```

Saudacao.java

```java
public class Saudacao {
    public static void main(String[] args) {
        saudacao("Fabio");
    }

    public static void saudacao(String nome) {
        System.out.println("Ola, " + nome + "!");
    }
}
```

Output

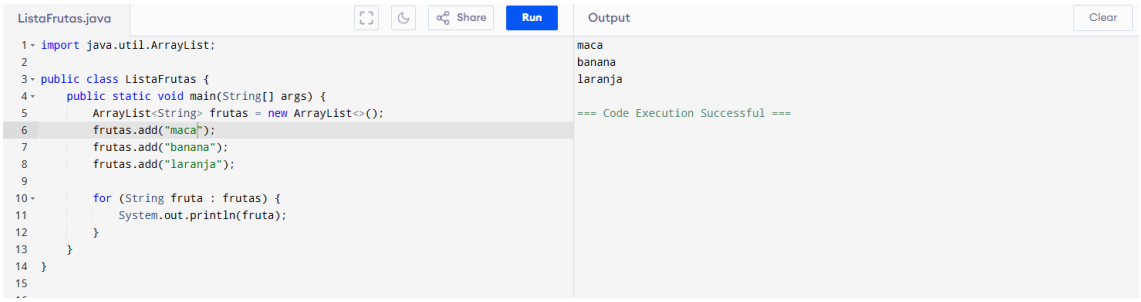
```
Ola, Fabio!

=== Code Execution Successful ===
```

- Criamos uma classe chamada **Saudacao**.
- No método **main()**, chamamos o método saudacao() passando "Fabio" como argumento.
- O método **saudacao(String nome)** recebe um parâmetro **nome** do tipo String.
- **System.out.println("Olá, " + nome + "!");** imprime uma mensagem personalizada.

7. Listas

Em Java, podemos utilizar **ArrayList** para representar listas mutáveis, como em Python. Veja o equivalente ao código fornecido:

```
import java.util.ArrayList;

public class ListaFrutas {
    public static void main(String[] args) {
        ArrayList<String> frutas = new ArrayList<>();
        frutas.add("maçã");
        frutas.add("banana");
        frutas.add("laranja");

        for (String fruta : frutas) {
            System.out.println(fruta);
        }
    }
}
```



- Importamos **ArrayList** da biblioteca java.util para trabalhar com listas dinâmicas.
- Criamos uma lista chamada **frutas** do tipo ArrayList<String>.
- Utilizamos **add()** para adicionar elementos à lista ("maçã", "banana" e "laranja").
- Utilizamos um **loop for-each** para percorrer cada elemento da lista e imprimi-lo.

---

8. Dicionários (MAPAS)

Em Java, o equivalente a um **dicionário** em Python é a estrutura **HashMap**, que permite armazenar pares chave-valor. Veja o equivalente ao código fornecido:

```
import java.util.HashMap;

public class MapaPessoa {
    public static void main(String[] args) {
        HashMap<String, Object> pessoa = new HashMap<>();
        pessoa.put("nome", "João");
        pessoa.put("idade", 30);
        pessoa.put("cidade", "São Paulo");

        System.out.println(pessoa.get("nome"));
    }
}
```

MapaPessoa.java

Share | Run | Output | Clear

```java
import java.util.HashMap;

public class MapaPessoa {
    public static void main(String[] args) {
        HashMap<String, Object> pessoa = new HashMap<>();
        pessoa.put("nome", "Joao");
        pessoa.put("idade", 30);
        pessoa.put("cidade", "São Paulo");

        System.out.println(pessoa.get("nome"));
    }
}
```

```
Joao

=== Code Execution Successful ===
```

- Importamos a classe **HashMap** da biblioteca java.util.
- Criamos um **HashMap<String, Object>**, onde as chaves são do tipo String e os valores podem ser de qualquer tipo (Object).
- Utilizamos **put()** para adicionar os pares chave-valor:
  - "nome" -> "João"
  - "idade" -> 30
  - "cidade" -> "São Paulo"
- Utilizamos **get("nome")** para acessar o valor associado à chave "nome" e imprimi-lo.

---

9. Manipulação de arquivos

Em Java, podemos manipular arquivos usando a classe **FileWriter** e **BufferedWriter** para criar e escrever arquivos, de forma semelhante ao Python. Veja o equivalente ao código fornecido:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ManipulacaoArquivo {
    public static void main(String[] args) {
        // Obtendo o caminho da pasta Downloads do usuário
        String downloadsPath = System.getProperty("user.home") + File.separator + "Downloads"
+ File.separator + "arquivo.txt";

        // Criando e escrevendo no arquivo
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(downloadsPath))) {
            writer.write("Olá, arquivo!");
            System.out.println("Arquivo criado e escrito com sucesso!");
        } catch (IOException e) {
            System.out.println("Ocorreu um erro ao manipular o arquivo: " + e.getMessage());
        }
    }
}
```

```java
package javaapplication12;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ManipulacaoArquivo {
    public static void main(String[] args) {
        // Obtendo o caminho da pasta Downloads do usuário
        String downloadsPath = System.getProperty("user.home") + File.separator + "Downloads" + File.separator + "arquivo.txt";

        // Criando e escrevendo no arquivo
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(downloadsPath))) {
            writer.write("Olá, arquivo!");
            System.out.println("Arquivo criado e escrito com sucesso!");
        } catch (IOException e) {
            System.out.println("Ocorreu um erro ao manipular o arquivo: " + e.getMessage());
        }
    }
}
```

```
Output - JavaApplication12 (run)
run:
Arquivo criado e escrito com sucesso!
BUILD SUCCESSFUL (total time: 0 seconds)
```

1. **Obter o caminho da pasta Downloads**:
   - System.getProperty("user.home") retorna o diretório do usuário.
   - File.separator adiciona o separador correto para o sistema operacional.
   - Criamos o caminho Downloads/arquivo.txt dinamicamente.
2. **Criar e escrever no arquivo**:
   - Usamos **FileWriter** para criar/escrever no arquivo.
   - **BufferedWriter** melhora a performance da escrita.
   - **try-with-resources** garante que o arquivo será fechado automaticamente após o uso.
3. **Tratamento de exceções**:
   - Se ocorrer algum erro ao criar/escrever no arquivo, **catch (IOException e)** captura a exceção e imprime a mensagem de erro.

---

10. Manipulando exceções

Em Java, usamos **try-catch** para capturar e tratar exceções, como a divisão por zero. Veja o equivalente ao código Python:

```java
public class TratamentoExcecao {
    public static void main(String[] args) {
        try {
            int resultado = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Erro: Divisão por zero.");
        }
    }
}
```

```
TratamentoExcecao.java                                   Share  Run
1  public class TratamentoExcecao {
2      public static void main(String[] args) {
3          try {
4              int resultado = 10 / 0;
5          } catch (ArithmeticException e) {
6              System.out.println("Erro: Divisao por zero.");
7          }
8      }
9  }
10
```

```
Output                                                   Clear
Erro: Divisao por zero.

=== Code Execution Successful ===
```

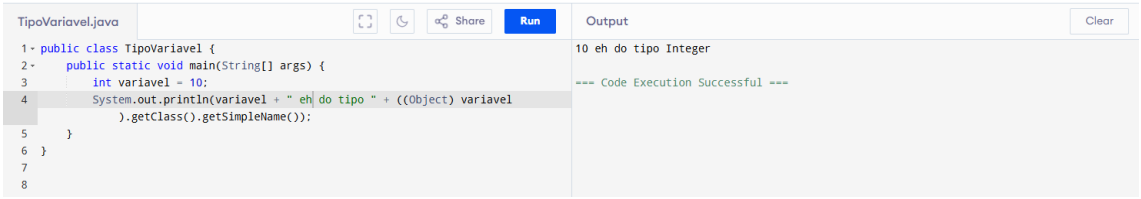
- O bloco **try** contém o código que pode gerar uma exceção.
- Tentamos dividir 10 / 0, o que em Java gera uma **ArithmeticException**.
- O bloco **catch (ArithmeticException e)** captura essa exceção e imprime uma mensagem personalizada.
- Se a exceção não fosse tratada, o programa encerraria com um erro.

---

11. Verificando tipo de variável

Em Java, não temos uma função direta equivalente ao type() do Python, pois Java é **fortemente tipado** e exige que o tipo de variável seja declarado explicitamente. No entanto, podemos usar **getClass().getSimpleName()** para obter o tipo de uma variável em tempo de execução. Veja o equivalente ao código fornecido:

```
public class TipoVariavel {
    public static void main(String[] args) {
        int variavel = 10;
        System.out.println(variavel + " é do tipo " + ((Object)
variavel).getClass().getSimpleName());
    }
}
```



- Criamos uma variável variavel do tipo **int**.
- Para obter o tipo da variável em tempo de execução, fazemos um **casting para Object** e chamamos **getClass().getSimpleName()**, que retorna o nome da classe do objeto.

---

12. Acessando elementos de uma lista (ArrayList)

Em Java, usamos **ArrayList** para representar listas dinâmicas, e podemos acessar seus elementos pelo índice, assim como em Python. Veja o equivalente ao código fornecido:

```
import java.util.ArrayList;

public class AcessoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);
        lista.add(5);

        System.out.println(lista.get(2)); // Acessa o terceiro elemento (índice 2)
    }
```

}

AcessoLista.java

```java
import java.util.ArrayList;

public class AcessoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);
        lista.add(5);

        System.out.println(lista.get(2)); // Acessa o terceiro elemento (índice 2
            )
    }
}
```

Output

```
3

=== Code Execution Successful ===
```

- Importamos **ArrayList** da biblioteca java.util para criar uma lista dinâmica.
- Criamos um **ArrayList<Integer>** e adicionamos os valores {1, 2, 3, 4, 5}.
- Utilizamos **get(2)** para acessar o terceiro elemento (3), já que a indexação em Java, assim como em Python, começa em **0**.

---

13. Modificando elementos de uma lista (ArrayList)

Em Java, podemos modificar elementos de um **ArrayList** diretamente pelo índice, assim como em Python. Veja o equivalente ao código fornecido:

```java
import java.util.ArrayList;

public class ModificaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);

        lista.set(1, 10); // Modifica o segundo elemento (índice 1) para 10

        System.out.println(lista); // Imprime [1, 10, 3, 4]
    }
}
```

ModificaLista.java

```java
import java.util.ArrayList;

public class ModificaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);

        lista.set(1, 10); // Modifica o segundo elemento (índice 1) para 10

        System.out.println(lista); // Imprime [1, 10, 3, 4]
    }
}
```

Output

```
[1, 10, 3, 4]

=== Code Execution Successful ===
```

- Criamos um **ArrayList<Integer>** e adicionamos os valores {1, 2, 3, 4}.
- Utilizamos **set(1, 10)** para substituir o segundo elemento (2) por 10.

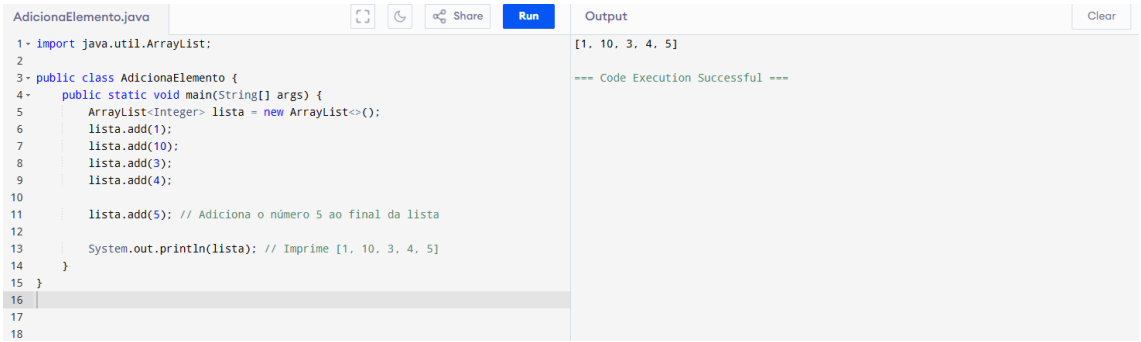
- **System.out.println(lista);** exibe a lista modificada.

---

14. Adicionando elementos a uma lista (ArrayList)

Em Java, o equivalente ao método append() do Python para adicionar um elemento ao final da lista é o método **add()** da classe **ArrayList**. Veja o código equivalente:

```
import java.util.ArrayList;

public class AdicionaElemento {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(10);
        lista.add(3);
        lista.add(4);

        lista.add(5); // Adiciona o número 5 ao final da lista

        System.out.println(lista); // Imprime [1, 10, 3, 4, 5]
    }
}
```



- Criamos um **ArrayList<Integer>** e adicionamos os valores {1, 10, 3, 4}.
- Utilizamos **add(5)** para inserir o valor 5 no final da lista.
- **System.out.println(lista);** exibe a lista modificada.

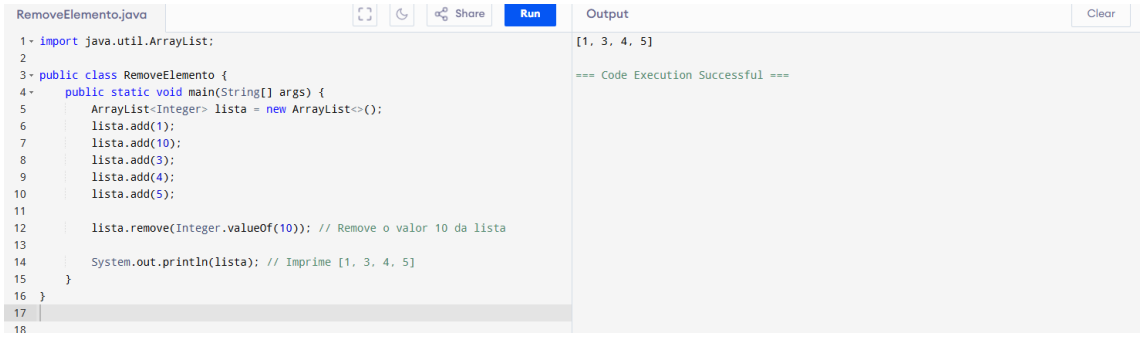
---

15. Removendo elementos de uma lista

Em Java, o equivalente ao método remove() do Python para excluir um elemento específico de uma lista é o método **remove(Object o)** da classe **ArrayList**. Veja o código equivalente:

```
import java.util.ArrayList;

public class RemoveElemento {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(10);
```

```
        lista.add(3);
        lista.add(4);
        lista.add(5);

        lista.remove(Integer.valueOf(10)); // Remove o valor 10 da lista

        System.out.println(lista); // Imprime [1, 3, 4, 5]
    }
}
```



- Criamos um **ArrayList<Integer>** e adicionamos os valores {1, 10, 3, 4, 5}.
- Utilizamos **remove(Integer.valueOf(10))** para remover o valor 10 da lista.
    - O método **remove()** pode remover por índice ou por valor. Como queremos remover o valor 10, usamos Integer.valueOf(10) para indicar que estamos removendo um **objeto Integer**, e não um índice.
- **System.out.println(lista);** exibe a lista modificada.

---

16. Ordenando uma lista

Em Java, o equivalente ao método sort() do Python para ordenar uma lista em ordem crescente é o método **Collections.sort()** da classe Collections. Veja o código equivalente:

```
import java.util.ArrayList;
import java.util.Collections;

public class OrdenaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(3);
        lista.add(1);
        lista.add(4);
        lista.add(5);

        Collections.sort(lista); // Ordena a lista em ordem crescente

        System.out.println(lista); // Imprime [1, 3, 4, 5]
    }
}
```

OrdenaLista.java

```java
import java.util.ArrayList;
import java.util.Collections;

public class OrdenaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(3);
        lista.add(1);
        lista.add(4);
        lista.add(5);

        Collections.sort(lista); // Ordena a lista em ordem crescente

        System.out.println(lista); // Imprime [1, 3, 4, 5]
    }
}
```

Output

```
[1, 3, 4, 5]

=== Code Execution Successful ===
```

- Criamos um **ArrayList<Integer>** e adicionamos os valores {3, 1, 4, 5} desordenados.
- Utilizamos **Collections.sort(lista)** para ordenar os elementos em ordem crescente.
- **System.out.println(lista);** exibe a lista ordenada.

---

17. Desempacotamento de listas (ArrayList e Arrays)

Em Java, não temos um desempacotamento direto como em Python (a, b, c = [1, 2, 3]), mas podemos atribuir elementos manualmente a variáveis de uma **lista (ArrayList)** ou de um **array**.

Exemplo usando ArrayList

```java
import java.util.ArrayList;

public class DesempacotamentoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);

        // Atribuindo manualmente os elementos a variáveis
        int a = lista.get(0);
        int b = lista.get(1);
        int c = lista.get(2);

        System.out.println(a + " " + b + " " + c); // Saída: 1 2 3
    }
}
```

DesempacotamentoLista.java

```java
import java.util.ArrayList;

public class DesempacotamentoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);

        // Atribuindo manualmente os elementos a variáveis
        int a = lista.get(0);
        int b = lista.get(1);
        int c = lista.get(2);

        System.out.println(a + " " + b + " " + c); // Saída: 1 2 3
    }
}
```

Output

```
1 2 3

=== Code Execution Successful ===
```

Exemplo usando um array

Se soubermos o tamanho fixo da lista, podemos usar um **array** para facilitar a atribuição:

```
public class DesempacotamentoArray {
    public static void main(String[] args) {
        int[] numeros = {1, 2, 3};

        // Desempacotando manualmente
        int a = numeros[0];
        int b = numeros[1];
        int c = numeros[2];

        System.out.println(a + " " + b + " " + c); // Saída: 1 2 3
    }
}
```

- Criamos uma lista (ArrayList<Integer>) ou um array (int[]).
- Atribuímos manualmente os valores aos elementos a, b e c usando **get(index)** para ArrayList ou índices diretos para arrays.
- Imprimimos os valores das variáveis separadas por espaço.

---

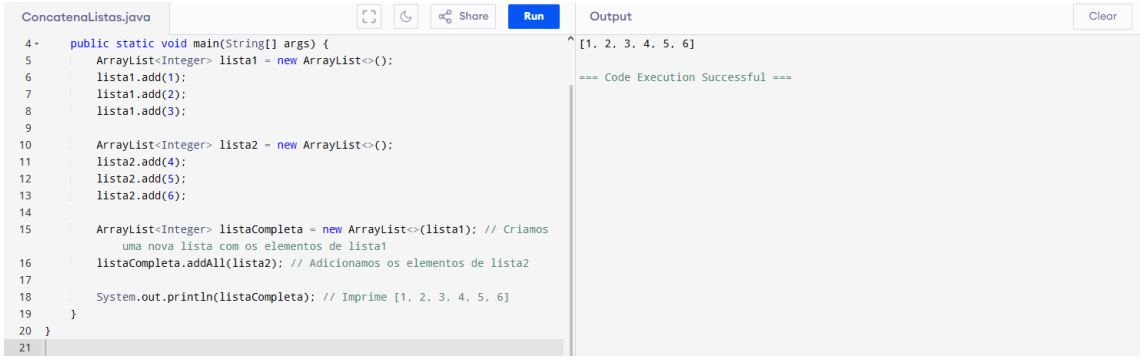
18. Concatenando listas (ArrayLis)

Em Java, a concatenação de listas não pode ser feita diretamente com o operador +, como em Python. Em vez disso, usamos o método **addAll()** da classe **ArrayList** para combinar duas listas.

Código equivalente em Java:

```
import java.util.ArrayList;

public class ConcatenaListas {
    public static void main(String[] args) {
        ArrayList<Integer> lista1 = new ArrayList<>();
        lista1.add(1);
        lista1.add(2);
        lista1.add(3);

        ArrayList<Integer> lista2 = new ArrayList<>();
        lista2.add(4);
        lista2.add(5);
        lista2.add(6);
```

```
        ArrayList<Integer> listaCompleta = new ArrayList<>(lista1); // Criamos uma nova lista com os elementos de lista1
        listaCompleta.addAll(lista2); // Adicionamos os elementos de lista2

        System.out.println(listaCompleta); // Imprime [1, 2, 3, 4, 5, 6]
    }
}
```



1. Criamos duas listas (lista1 e lista2) e adicionamos os elementos.
2. Criamos uma terceira lista **listaCompleta**, inicializada com os elementos de lista1.
3. Utilizamos **addAll(lista2)** para adicionar os elementos da segunda lista à listaCompleta.
4. Imprimimos a lista concatenada.

---

19. Tuplas

Em Java, não há um tipo nativo equivalente às **tuplas** do Python, pois as listas em Java (ArrayList) são mutáveis. No entanto, podemos representar tuplas de diferentes maneiras:

1. Usando List.of() (Lista Imutável - Java 9+)

A partir do **Java 9**, podemos criar listas imutáveis usando List.of(), que se comportam de forma semelhante a uma tupla em Python.

```
import java.util.List;

public class TuplaImutavel {
    public static void main(String[] args) {
        List<Integer> tupla = List.of(1, 2, 3); // Lista imutável

        System.out.println(tupla.get(0)); // Acessa o primeiro elemento (1)
    }
}
```

TuplaImutavel.java

```java
import java.util.List;

public class TuplaImutavel {
    public static void main(String[] args) {
        List<Integer> tupla = List.of(1, 2, 3); // Lista imutável

        System.out.println(tupla.get(0)); // Acessa o primeiro elemento (1)
    }
}
```

Output

```
1

=== Code Execution Successful ===
```

🔻 **Vantagem**: List.of() cria uma lista imutável, impedindo modificações, assim como as tuplas em Python.
🔻 **Desvantagem**: Ainda é uma lista, não uma estrutura nomeada como uma tupla.

---

2. Criando uma Tupla com record (Java 14+)

Desde **Java 14**, podemos usar record para criar uma estrutura semelhante a uma tupla, onde os valores são imutáveis:

```java
public record Tupla(int primeiro, int segundo, int terceiro) {}

public class ExemploTupla {
    public static void main(String[] args) {
        Tupla tupla = new Tupla(1, 2, 3);

        System.out.println(tupla.primeiro()); // Acessa o primeiro elemento (1)
    }
}
```

🔻 **Vantagem**: record permite criar objetos **imutáveis**, semelhantes a tuplas nomeadas.
🔻 **Desvantagem**: Requer definir a estrutura antecipadamente, diferente das tuplas dinâmicas de Python.

---

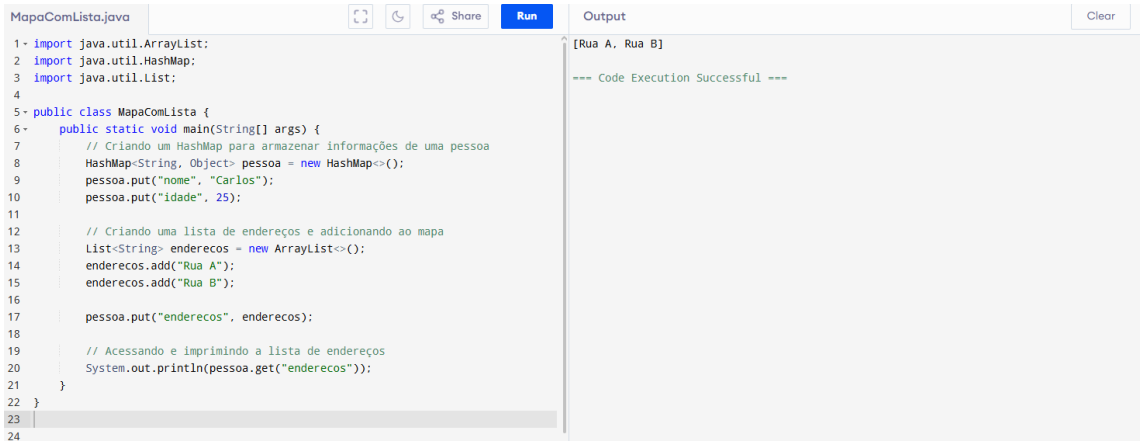
20. Dicionários com listas (HashMap)

Em Java, o equivalente a um **dicionário (dict)** em Python é a estrutura **HashMap**, que permite armazenar pares **chave-valor**. Como valores, podemos armazenar **listas (ArrayList)**, assim como em Python.

Código equivalente em Java:

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class MapaComLista {
    public static void main(String[] args) {
        // Criando um HashMap para armazenar informações de uma pessoa
        HashMap<String, Object> pessoa = new HashMap<>();
        pessoa.put("nome", "Carlos");
        pessoa.put("idade", 25);
```

```
        // Criando uma lista de endereços e adicionando ao mapa
        List<String> enderecos = new ArrayList<>();
        enderecos.add("Rua A");
        enderecos.add("Rua B");

        pessoa.put("enderecos", enderecos);

        // Acessando e imprimindo a lista de endereços
        System.out.println(pessoa.get("enderecos"));
    }
}
```



1. Criamos um **HashMap<String, Object>**, onde as chaves são String e os valores podem ser de qualquer tipo (Object).
2. Adicionamos "nome" e "idade" ao mapa.
3. Criamos uma **ArrayList<String>** chamada enderecos e adicionamos "Rua A" e "Rua B".
4. Inserimos essa lista no mapa sob a chave "enderecos".
5. Usamos **pessoa.get("enderecos")** para acessar e imprimir a lista de endereços.

Neste passo a passo, exploramos os conceitos fundamentais de Python, incluindo operações matemáticas, estruturas condicionais, loops, funções, listas, dicionários e manipulação de arquivos.

Com esses conceitos, você já pode começar a escrever scripts mais complexos e resolver problemas de programação de forma mais eficiente.

Estes exemplos podem ser usados como base para entender e praticar a linguagem Python.

Experimente modificar os exemplos e criar suas próprias variações para se aprofundar no aprendizado!

Codigos para Praticar

```
nome = "fabio";
print(nome);
```

```java
# Exemplo 1: Imprimir uma mensagem simples
public class OlaMundo {
    public static void main(String[] args) {
        System.out.println("Olá, mundo!");
    }
}

# Exemplo 2: Operações matemáticas básicas
public class OperacoesMatematicas {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;

        System.out.println(a + b); // Soma
        System.out.println(a - b); // Subtração
        System.out.println(a * b); // Multiplicação
        System.out.println((double) a / b); // Divisão com conversão para double
    }
}

# Exemplo 3: Usando estruturas condicionais (if-else)
public class VerificaIdade {
    public static void main(String[] args) {
        int idade = 18;

        if (idade >= 18) {
            System.out.println("Você é maior de idade.");
        } else {
            System.out.println("Você é menor de idade.");
        }
    }
}

# Exemplo 4: Loops (for loop)
public class LoopFor {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println(i);
        }
    }
}

# Exemplo 5: Loops (while loop)
public class LoopWhile {
    public static void main(String[] args) {
        int contador = 0;

        while (contador < 5) {
            System.out.println(contador);
            contador++;
        }
    }
}

# Exemplo 6: Definindo funções
public class Saudacao {
```

```
    public static void main(String[] args) {
        saudacao("Maria");
    }

    public static void saudacao(String nome) {
        System.out.println("Olá, " + nome + "!");
    }
}
```

Exemplo 7: Listas

```
import java.util.ArrayList;

public class ListaFrutas {
    public static void main(String[] args) {
        ArrayList<String> frutas = new ArrayList<>();
        frutas.add("maçã");
        frutas.add("banana");
        frutas.add("laranja");

        for (String fruta : frutas) {
            System.out.println(fruta);
        }
    }
}
```

Exemplo 8: Dicionários

```
import java.util.HashMap;

public class MapaPessoa {
    public static void main(String[] args) {
        // Criando um HashMap para armazenar informações de uma pessoa
        HashMap<String, Object> pessoa = new HashMap<>();
        pessoa.put("nome", "João");
        pessoa.put("idade", 30);
        pessoa.put("cidade", "São Paulo");

        // Acessando e imprimindo o valor da chave "nome"
        System.out.println(pessoa.get("nome"));
    }
}
```

Exemplo 9: Manipulação de arquivos

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ManipulacaoArquivo {
    public static void main(String[] args) {
        // Obtendo o caminho da pasta Downloads do usuário
        String downloadsPath = System.getProperty("user.home") + File.separator + "Downloads"
+ File.separator + "arquivo.txt";

        // Criando e escrevendo no arquivo
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(downloadsPath))) {
            writer.write("Olá, arquivo!");
            System.out.println("Arquivo criado e escrito com sucesso!");
```

```
        } catch (IOException e) {
            System.out.println("Ocorreu um erro ao manipular o arquivo: " + e.getMessage());
        }
    }
}
```

Exemplo 10: Manipulando exceções

```
public class TratamentoExcecao {
    public static void main(String[] args) {
        try {
            int resultado = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Erro: Divisão por zero.");
        }
    }
}
```

Exemplo 11: Verificando tipo de variável

```
public class TipoVariavel {
    public static void main(String[] args) {
        int variavel = 10;
        System.out.println(((Object) variavel).getClass().getSimpleName());
    }
}
```

Exemplo 12: Acessando elementos de uma lista

```
import java.util.ArrayList;

public class AcessoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);

        System.out.println(lista.get(2)); // Acessa o terceiro elemento (índice 2)
    }
}
```

Exemplo 13: Modificando elementos de uma lista

```
import java.util.ArrayList;

public class ModificaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(2);
        lista.add(3);
        lista.add(4);

        lista.set(1, 10); // Modifica o segundo elemento (índice 1) para 10

        System.out.println(lista); // Imprime [1, 10, 3, 4]
    }
}
```

Exemplo 14: Adicionando elementos a uma lista
```
import java.util.ArrayList;

public class AdicionaElemento {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(10);
        lista.add(3);
        lista.add(4);

        lista.add(5); // Adiciona o número 5 ao final da lista

        System.out.println(lista); // Imprime [1, 10, 3, 4, 5]
    }
}
```

Exemplo 15: Removendo elementos de uma lista
```
import java.util.ArrayList;

public class RemoveElemento {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(1);
        lista.add(10);
        lista.add(3);
        lista.add(4);
        lista.add(5);

        lista.remove(Integer.valueOf(10)); // Remove o valor 10 da lista

        System.out.println(lista); // Imprime [1, 3, 4, 5]
    }
}
```

Exemplo 16: Ordenando uma lista
```
import java.util.ArrayList;
import java.util.Collections;

public class OrdenaLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>();
        lista.add(3);
        lista.add(1);
        lista.add(4);
        lista.add(5);

        Collections.sort(lista); // Ordena a lista em ordem crescente

        System.out.println(lista); // Imprime [1, 3, 4, 5]
    }
}
```

Exemplo 17: Desempacotamento de listas
```
public class DesempacotamentoArray {
    public static void main(String[] args) {
        int[] numeros = {1, 2, 3};
```

```
        // Desempacotando manualmente
        int a = numeros[0];
        int b = numeros[1];
        int c = numeros[2];

        System.out.println(a + " " + b + " " + c); // Saída: 1 2 3
    }
}
```

```
import java.util.ArrayList;
import java.util.Arrays;

public class DesempacotamentoLista {
    public static void main(String[] args) {
        ArrayList<Integer> lista = new ArrayList<>(Arrays.asList(1, 2, 3));

        // Atribuindo manualmente os elementos a variáveis
        int a = lista.get(0);
        int b = lista.get(1);
        int c = lista.get(2);

        System.out.println(a + " " + b + " " + c); // Saída: 1 2 3
    }
}
```

Exemplo 18: Concatenando listas

```
import java.util.ArrayList;

public class ConcatenaListas {
    public static void main(String[] args) {
        ArrayList<Integer> lista1 = new ArrayList<>();
        lista1.add(1);
        lista1.add(2);
        lista1.add(3);

        ArrayList<Integer> lista2 = new ArrayList<>();
        lista2.add(4);
        lista2.add(5);
        lista2.add(6);

        ArrayList<Integer> listaCompleta = new ArrayList<>(lista1); // Criamos uma nova lista com
os elementos de lista1
        listaCompleta.addAll(lista2); // Adicionamos os elementos de lista2

        System.out.println(listaCompleta); // Imprime [1, 2, 3, 4, 5, 6]
    }
}
```

Exemplo 19: Tuplas

```
import java.util.List;

public class TuplaImutavel {
    public static void main(String[] args) {
        List<Integer> tupla = List.of(1, 2, 3); // Lista imutável
```

```
        System.out.println(tupla.get(0)); // Acessa o primeiro elemento (1)
    }
}


public record Tupla(int primeiro, int segundo, int terceiro) {}

public class ExemploTupla {
    public static void main(String[] args) {
        Tupla tupla = new Tupla(1, 2, 3);

        System.out.println(tupla.primeiro()); // Acessa o primeiro elemento (1)
    }
}
```

Exemplo 20: Dicionários com listas

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class MapaComLista {
    public static void main(String[] args) {
        // Criando um HashMap para armazenar informações de uma pessoa
        HashMap<String, Object> pessoa = new HashMap<>();
        pessoa.put("nome", "Carlos");
        pessoa.put("idade", 25);

        // Criando uma lista de endereços e adicionando ao mapa
        List<String> enderecos = new ArrayList<>();
        enderecos.add("Rua A");
        enderecos.add("Rua B");

        pessoa.put("enderecos", enderecos);

        // Acessando e imprimindo a lista de endereços
        System.out.println(pessoa.get("enderecos"));
    }
}
```

Exemplo 21: Definindo funções com parâmetros

```
public class Calculadora {
    // Método que retorna a soma de dois números
    public static int soma(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int resultado = soma(5, 10);
        System.out.println(resultado); // Saída: 15
    }
}
```

Exemplo 22: Função com valor de retorno

```
public class Calculadora {
    // Método que retorna o quadrado de um número
```

```
    public static int quadrado(int x) {
        return x * x;
    }

    public static void main(String[] args) {
        System.out.println(quadrado(4)); // Saída: 16
    }
}
```

Exemplo 23: Funções com argumentos padrões

```
public class Saudacao {
    // Método com argumento
    public static void saudacao(String nome) {
        System.out.println("Olá, " + nome + "!");
    }

    // Método sem argumento (valor padrão "Visitante")
    public static void saudacao() {
        saudacao("Visitante");
    }

    public static void main(String[] args) {
        saudacao("Ana");   // Saída: Olá, Ana!
        saudacao();        // Saída: Olá, Visitante!
    }
}
```

Exemplo 24: Função recursiva

```
public class Fatorial {
    // Método recursivo para calcular o fatorial
    public static int fatorial(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n * fatorial(n - 1);
        }
    }

    public static void main(String[] args) {
        System.out.println(fatorial(5)); // Saída: 120
    }
}
```

Exemplo 25: Lambdas (funções anônimas)

```
import java.util.function.BiFunction;

public class LambdaSoma {
    public static void main(String[] args) {
        // Expressão lambda que recebe dois inteiros e retorna sua soma
        BiFunction<Integer, Integer, Integer> soma = (a, b) -> a + b;

        System.out.println(soma.apply(3, 4)); // Saída: 7
    }
}
```

Exemplo 26: List comprehension

```
import java.util.List;
```

```
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class ListaQuadrados {
    public static void main(String[] args) {
        // Criando uma lista de quadrados de 0 a 4
        List<Integer> quadrados = IntStream.range(0, 5) // Gera os números 0 a 4
                                    .map(x -> x * x) // Calcula o quadrado de cada número
                                    .boxed() // Converte de int para Integer
                                    .collect(Collectors.toList()); // Coleta os resultados em uma lista

        System.out.println(quadrados); // Saída: [0, 1, 4, 9, 16]
    }
}
```

Se quiser evitar o uso da Stream API, podemos criar a lista manualmente usando um loop for:

```
import java.util.ArrayList;
import java.util.List;

public class ListaQuadrados {
    public static void main(String[] args) {
        List<Integer> quadrados = new ArrayList<>();

        for (int x = 0; x < 5; x++) {
            quadrados.add(x * x);
        }

        System.out.println(quadrados); // Saída: [0, 1, 4, 9, 16]
    }
}
```

Exemplo 27: Funções de map

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class MapeamentoLista {
    public static void main(String[] args) {
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4);

        // Aplicando uma função para calcular o quadrado de cada número
        List<Integer> quadrados = numeros.stream()
                                    .map(x -> x * x) // Aplica a transformação (elevar ao quadrado)
                                    .collect(Collectors.toList()); // Converte o resultado para lista

        System.out.println(quadrados); // Saída: [1, 4, 9, 16]
    }
}
```

Se quiser evitar o uso de Stream API, podemos transformar os valores manualmente com um for:

```
import java.util.ArrayList;
import java.util.List;

public class MapeamentoLista {
    public static void main(String[] args) {
        List<Integer> numeros = List.of(1, 2, 3, 4);
        List<Integer> quadrados = new ArrayList<>();

        for (int x : numeros) {
            quadrados.add(x * x);
        }

        System.out.println(quadrados); // Saída: [1, 4, 9, 16]
    }
}
```

Exemplo 28: Função de filtro

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class FiltraLista {
    public static void main(String[] args) {
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4);

        // Filtrando os números pares
        List<Integer> pares = numeros.stream()
                                .filter(x -> x % 2 == 0) // Mantém apenas os números pares
                                .collect(Collectors.toList()); // Converte o resultado em uma lista

        System.out.println(pares); // Saída: [2, 4]
    }
}
```

Se preferirmos evitar o uso de Stream API, podemos filtrar os valores manualmente com um for:

```
import java.util.ArrayList;
import java.util.List;

public class FiltraLista {
    public static void main(String[] args) {
        List<Integer> numeros = List.of(1, 2, 3, 4);
        List<Integer> pares = new ArrayList<>();

        for (int x : numeros) {
            if (x % 2 == 0) {
                pares.add(x);
            }
        }

        System.out.println(pares); // Saída: [2, 4]
```

```
    }
}
```

Exemplo 29: Função reduce

Em **Python**, usamos **reduce()** da biblioteca functools para reduzir uma lista a um único valor aplicando uma função cumulativa. Em **Java**, a melhor forma de fazer isso é utilizando **Stream API** e o método **reduce()**, introduzido no **Java 8**.

```
import java.util.Arrays;
import java.util.List;

public class ReduceLista {
    public static void main(String[] args) {
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4);

        // Reduzindo a lista somando todos os elementos
        int somaTotal = numeros.stream()
                             .reduce(0, (x, y) -> x + y); // Redução acumulativa

        System.out.println(somaTotal); // Saída: 10
    }
}
```

Se preferirmos evitar o uso de Stream API, podemos calcular a soma manualmente com um for:

```
import java.util.List;

public class ReduceLista {
    public static void main(String[] args) {
        List<Integer> numeros = List.of(1, 2, 3, 4);
        int somaTotal = 0;

        for (int num : numeros) {
            somaTotal += num;
        }

        System.out.println(somaTotal); // Saída: 10
    }
}
```

Exemplo 30: Compreensão de dicionários

Em **Python**, usamos **dictionary comprehension** para criar um dicionário de forma concisa ({x: x ** 2 for x in range(5)}).
Em **Java**, podemos obter um comportamento semelhante usando **Stream API**, **Collectors.toMap()**, ou um **loop tradicional** para preencher um HashMap.

```
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
```

```
public class MapaQuadrados {
    public static void main(String[] args) {
        // Criando um HashMap onde as chaves são números e os valores são seus quadrados
        Map<Integer, Integer> dicionario = IntStream.range(0, 5) // Gera os números de 0 a 4
                .boxed() // Converte de int para Integer
                .collect(Collectors.toMap(x -> x, x -> x * x)); // Mapeia chave -> valor

        System.out.println(dicionario); // Saída: {0=0, 1=1, 2=4, 3=9, 4=16}
    }
}
```

Exemplo 31: Acessando caracteres de uma string

```
public class AcessoString {
    public static void main(String[] args) {
        String texto = "Python";
        System.out.println(texto.charAt(0)); // Acessa o primeiro caractere ('P')
    }
}
```

Exemplo 32: Fatiamento de strings

Em **Python**, usamos texto[1:4] para obter uma parte da string (**fatiamento**).
Em **Java**, utilizamos o método **substring(inicio, fim)**, onde:

- inicio → Índice inicial (**inclusivo**)
- fim → Índice final (**exclusivo**)

```
public class FatiamentoString {
    public static void main(String[] args) {
        String texto = "Python";
        System.out.println(texto.substring(1, 4)); // Obtém "yth"
    }
}
```

Exemplo 33: Concatenando strings

Em **Python**, usamos o operador + para concatenar strings (nome + " " + sobrenome).
Em **Java**, a concatenação de strings também pode ser feita com **+**, mas também podemos usar **concat()** ou **StringBuilder** para maior eficiência.

Código equivalente em Java (usando +):

```
public class ConcatenaString {
    public static void main(String[] args) {
        String nome = "Ana";
        String sobrenome = "Silva";
        String nomeCompleto = nome + " " + sobrenome;

        System.out.println(nomeCompleto); // Saída: Ana Silva
    }
}
```

Exemplo 34: Métodos de string

```
public class ConverterMaiusculas {
```

```
    public static void main(String[] args) {
        String frase = "olá mundo";
        System.out.println(frase.toUpperCase()); // Saída: OLÁ MUNDO
    }
}
```

Exemplo 35: Substituindo palavras em uma string

```
public class SubstituirTexto {
    public static void main(String[] args) {
        String texto = "Eu gosto de Python";
        String novoTexto = texto.replace("Python", "Java");

        System.out.println(novoTexto); // Saída: Eu gosto de Java
    }
}
```

Exemplo 36: Dividindo uma string

```
import java.util.Arrays;

public class DividirString {
    public static void main(String[] args) {
        String texto = "Python é ótimo";
        String[] palavras = texto.split(" "); // Divide a string pelos espaços

        System.out.println(Arrays.toString(palavras)); // Saída: [Python, é, ótimo]
    }
}
```

Exemplo 37: Verificando se uma substring está presente

```
public class VerificaSubstring {
    public static void main(String[] args) {
        String texto = "Python é ótimo";

        System.out.println(texto.contains("Python")); // Saída: true
    }
}
```

Exemplo 38: Função join para concatenar lista de strings

Em **Python**, usamos **" ".join(lista)** para unir elementos de uma lista em uma única string. Em **Java**, fazemos isso com **String.join(" ", lista)**, que concatena os elementos com um separador específico.

```
import java.util.List;
import java.util.StringJoiner;

public class JuntarStrings {
    public static void main(String[] args) {
        List<String> palavras = List.of("Python", "é", "legal");

        String frase = String.join(" ", palavras);

        System.out.println(frase); // Saída: Python é legal
    }
}
```

Exemplo 39: Usando o operador "in" com listas

Em **Python**, usamos in para verificar se um elemento está presente em uma lista (3 in numeros).
Em **Java**, usamos o método **contains()** da classe **List** (ArrayList).

```
import java.util.List;

public class VerificaElemento {
    public static void main(String[] args) {
        List<Integer> numeros = List.of(1, 2, 3, 4, 5);

        System.out.println(numeros.contains(3)); // Saída: true
    }
}
```

Exemplo 40: Manipulando conjuntos

```
import java.util.HashSet;
import java.util.Set;

public class ManipulaConjunto {
    public static void main(String[] args) {
        Set<Integer> conjunto = new HashSet<>();
        conjunto.add(1);
        conjunto.add(2);
        conjunto.add(3);

        conjunto.add(4); // Adiciona um novo elemento ao conjunto

        System.out.println(conjunto); // Saída: [1, 2, 3, 4] (A ordem pode variar)
    }
}
```

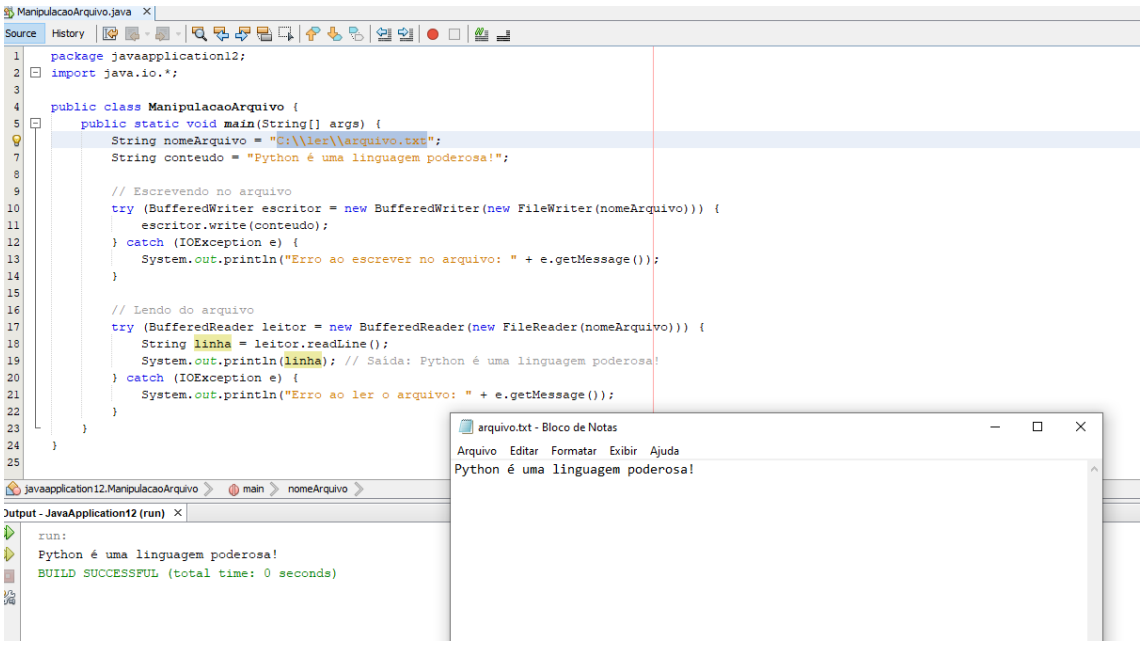
Exemplo 41: Diferença entre listas e conjuntos

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class ListaParaConjunto {
    public static void main(String[] args) {
        List<Integer> lista = Arrays.asList(1, 2, 3, 3, 4);

        Set<Integer> conjunto = new HashSet<>(lista); // Converte a lista para um conjunto
(remove duplicatas)

        System.out.println(conjunto); // Saída: [1, 2, 3, 4] (Ordem pode variar)
    }
}
```

Exemplo 42: Criando e manipulando arquivos

```
import java.io.*;

public class ManipulacaoArquivo {
    public static void main(String[] args) {
        String nomeArquivo = " C:\\ler\\arquivo.txt";
        String conteudo = "Python é uma linguagem poderosa!";
```

```
        // Escrevendo no arquivo
        try (BufferedWriter escritor = new BufferedWriter(new FileWriter(nomeArquivo))) {
            escritor.write(conteudo);
        } catch (IOException e) {
            System.out.println("Erro ao escrever no arquivo: " + e.getMessage());
        }

        // Lendo do arquivo
        try (BufferedReader leitor = new BufferedReader(new FileReader(nomeArquivo))) {
            String linha = leitor.readLine();
            System.out.println(linha); // Saída: Python é uma linguagem poderosa!
        } catch (IOException e) {
            System.out.println("Erro ao ler o arquivo: " + e.getMessage());
        }
    }
}
```



Exemplo 43: Manipulando diretórios

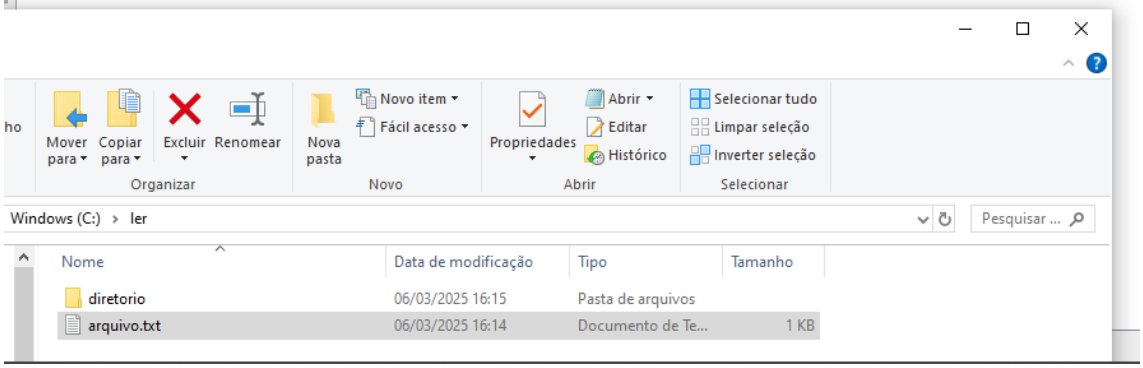
Em **Python**, usamos os.mkdir("meu_diretorio") para criar um diretório.
Em **Java**, usamos a classe **File** e o método **mkdir()** para criar um diretório.

```
import java.io.File;

public class CriarDiretorio {
    public static void main(String[] args) {
        File diretorio = new File("meu_diretorio");

        if (diretorio.mkdir()) {
            System.out.println("Diretório criado!");
        } else {
            System.out.println("O diretório já existe ou não pôde ser criado.");
        }
    }
}
```

```java
import java.io.File;

public class CriarDiretorio {
    public static void main(String[] args) {
        File diretorio = new File("C:\\ler\\diretorio");

        if (diretorio.mkdir()) {
            System.out.println("Diretório criado!");
        } else {
            System.out.println("O diretório já existe ou não pôde ser criado.");
        }
    }
}
```

javaapplication12.CriarDiretorio > main > diretorio

```
run:
Diretório criado!
BUILD SUCCESSFUL (total time: 0 seconds)
```



Exemplo 44: Verificando se um arquivo ou diretório existe

```java
import java.io.File;

public class VerificaArquivo {
    public static void main(String[] args) {
        File arquivo = new File("meuarquivo.txt");

        if (arquivo.exists()) {
            System.out.println("true"); // Arquivo existe
        } else {
            System.out.println("false"); // Arquivo não existe
        }
    }
}
```

```java
import java.io.File;

public class VerificaArquivo {
    public static void main(String[] args) {
        File arquivo = new File("C:\\ler\\arquivo.txt");

        if (arquivo.exists()) {
            System.out.println("true"); // Arquivo existe
        } else {
            System.out.println("false"); // Arquivo não existe
        }
    }
}
```

javaapplication12.VerificaArquivo > main > arquivo

tput - JavaApplication12 (run)

```
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Exemplo 45: Listando arquivos em um diretório

```java
import java.io.File;

public class ListarArquivos {
    public static void main(String[] args) {
        File diretorioAtual = new File("."); // Diretório atual
        File[] arquivos = diretorioAtual.listFiles(); // Lista arquivos e diretórios

        if (arquivos != null) {
            for (File arquivo : arquivos) {
                System.out.println(arquivo.getName()); // Exibe os nomes dos arquivos
            }
        } else {
            System.out.println("Não foi possível listar os arquivos.");
        }
    }
}
```

```java
import java.io.File;

public class ListarArquivos {
    public static void main(String[] args) {
        File diretorioAtual = new File("."); // Diretório atual
        File[] arquivos = diretorioAtual.listFiles(); // Lista arquivos e diretórios

        if (arquivos != null) {
            for (File arquivo : arquivos) {
                System.out.println(arquivo.getName()); // Exibe os nomes dos arquivos
            }
        } else {
            System.out.println("Não foi possível listar os arquivos.");
        }
    }
}
```

ListarArquivos > main > arquivos >

utput - JavaApplication12 (run) ×

```
run:
arquivo.txt
build
build.xml
manifest.mf
meuarquivo.txt
nbproject
src
BUILD SUCCESSFUL (total time: 0 seconds)
```

Exemplo 46: Utilizando a biblioteca math

```java
public class RaizQuadrada {
    public static void main(String[] args) {
        double resultado = Math.sqrt(16);
        System.out.println(resultado); // Saída: 4.0
    }
}
```

Exemplo 47: Gerando números aleatórios

Em **Python**, usamos random.randint(1, 10) para gerar um número aleatório inteiro entre 1 e 10 (inclusive).
Em **Java**, podemos usar a classe **Random** ou **Math.random()** para obter o mesmo comportamento.

```java
import java.util.Random;

public class NumeroAleatorio {
    public static void main(String[] args) {
        Random random = new Random();
        int numero = random.nextInt(10) + 1; // Gera um número entre 1 e 10 (inclusive)

        System.out.println(numero);
    }
}
```

Exemplo 48: Trabalhando com data e hora

```java
import java.time.LocalDateTime;

public class DataHoraAtual {
    public static void main(String[] args) {
        LocalDateTime hoje = LocalDateTime.now(); // Obtém a data e hora atual
```

```
        System.out.println(hoje); // Exibe a data e hora atual
    }
}
```

Alternativa Formatando a Saída (DateTimeFormatter**)**

Se quisermos exibir a data e hora em um formato personalizado, usamos

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class DataHoraFormatada {
    public static void main(String[] args) {
        LocalDateTime hoje = LocalDateTime.now();
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

        String dataFormatada = hoje.format(formato);
        System.out.println(dataFormatada); // Exibe data e hora no formato dd/MM/yyyy HH:mm:ss
    }
}
```

Exemplo 49: Tratamento de erros com try-except

Em **Python**, usamos input() para obter a entrada do usuário e int() para convertê-la para número. Se a entrada for inválida, capturamos o erro com except ValueError.
Em **Java**, usamos **Scanner** para capturar a entrada e **try-catch** para tratar exceções (NumberFormatException).

```
import java.util.Scanner;

public class EntradaUsuario {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Digite um número: ");
            int numero = Integer.parseInt(scanner.nextLine()); // Converte a entrada para int
            System.out.println("Número digitado: " + numero);
        } catch (NumberFormatException e) {
            System.out.println("Erro: Entrada inválida!");
        } finally {
            scanner.close(); // Fecha o Scanner para evitar vazamento de recursos
        }
    }
}
```

Exemplo 50: Função de ajuda (help)

Em **Python**, usamos help(print) para exibir a documentação da função print().
Em **Java**, há **duas formas principais** de obter documentação sobre métodos:

1. **Usando javap no terminal (Linha de Comando)**
2. **Usando Javadoc nos ambientes de desenvolvimento (IDE como IntelliJ, NetBeans ou Eclipse)**

Se você estiver executando Java via terminal, pode usar o comando:

javap -classpath rt.jar java.io.PrintStream

Esse comando exibe todos os métodos da classe **PrintStream**, incluindo print(), pois em Java, System.out.print() pertence a essa classe.

Se estiver usando uma **IDE**, você pode obter a documentação de print() diretamente:

✅ **No IntelliJ IDEA:**

- **Coloque o cursor sobre print() e pressione Ctrl + Q (Windows/Linux) ou Cmd + Q (Mac).**

✅ **No Eclipse:**

- **Coloque o cursor sobre print() e pressione F2 para exibir o Javadoc.**

✅ **No NetBeans:**

- **Passe o mouse sobre print() para visualizar a documentação.**

Exemplo de Código em Java para Testar print()

```
public class TestePrint {
    public static void main(String[] args) {
        System.out.print("Olá, mundo!"); // Exibe texto sem quebra de linha
        System.out.println(" Olá, mundo!"); // Exibe texto com quebra de linha
    }
}
```

Dicas:

- **Em Python**, usamos help(print).
- **Em Java**, usamos **javap no terminal** ou **Javadoc nas IDEs** (Ctrl + Q no IntelliJ, F2 no Eclipse).
- **O método print() faz parte da classe PrintStream, acessível via System.out.print()**.

Projeto

Esse projeto representa um **Gerenciador de Tarefas** onde é possível adicionar, listar e remover tarefas. O código está bem comentado para facilitar a explicação na faculdade.

Funcionalidades do projeto:

- Adicionar uma nova tarefa
- Listar todas as tarefas
- Remover uma tarefa pelo índice

Explicação:

Criação da classe GerenciadorDeTarefas

- Possui um atributo tarefas, que armazena as tarefas em uma **ArrayList**.

Método adicionarTarefa(String descricao)

- Adiciona uma nova tarefa à lista e exibe uma mensagem de confirmação.

Método listarTarefas()

- Percorre a lista e exibe as tarefas numeradas.
- Se a lista estiver vazia, exibe uma mensagem informando.

Método removerTarefa(int indice)

- Remove uma tarefa com base no índice informado pelo usuário.
- Garante que o índice seja válido antes de remover.

Testando a classe:

- Criamos uma instância de GerenciadorDeTarefas.
- Adicionamos três tarefas de exemplo.
- Listamos as tarefas cadastradas.
- Removemos uma tarefa específica.
- Listamos novamente para verificar a remoção.

Codigo

```
package com.educaciencia.fastcode;
```

```java
import java.util.ArrayList;
import java.util.List;

public class GerenciadorDeTarefas {
    /**
     * Classe para gerenciar uma lista de tarefas simples.
     * Permite adicionar, listar e remover tarefas.
     */

    private List<String> tarefas; // Lista que armazenará as tarefas

    /**
     * Construtor que inicializa a lista de tarefas vazia.
     */
    public GerenciadorDeTarefas() {
        this.tarefas = new ArrayList<>();
    }

    /**
     * Adiciona uma nova tarefa à lista.
     * @param descricao - Descrição da tarefa.
     */
    public void adicionarTarefa(String descricao) {
        tarefas.add(descricao);
        System.out.println("Tarefa adicionada: " + descricao);
    }

    /**
     * Lista todas as tarefas cadastradas.
     */
    public void listarTarefas() {
        if (tarefas.isEmpty()) {
            System.out.println("Nenhuma tarefa cadastrada.");
        } else {
            System.out.println("\nLista de Tarefas:");
            for (int i = 0; i < tarefas.size(); i++) {
                System.out.println((i + 1) + ". " + tarefas.get(i));
            }
        }
    }

    /**
     * Remove uma tarefa da lista com base no índice informado.
     * @param indice - Índice da tarefa a ser removida (começa de 1 para o usuário).
     */
    public void removerTarefa(int indice) {
```

```
        if (indice >= 1 && indice <= tarefas.size()) {
            String tarefaRemovida = tarefas.remove(indice - 1);
            System.out.println("Tarefa removida: " + tarefaRemovida);
        } else {
            System.out.println("Índice inválido! Tente novamente.");
        }
    }

    public static void main(String[] args) {
        // Criando uma instância da classe
        GerenciadorDeTarefas gerenciador = new GerenciadorDeTarefas();

        // Adicionando algumas tarefas
        gerenciador.adicionarTarefa("Estudar para a prova de matemática");
        gerenciador.adicionarTarefa("Ler um artigo sobre Inteligência Artificial");
        gerenciador.adicionarTarefa("Fazer exercícios físicos");

        // Listando as tarefas
        gerenciador.listarTarefas();

        // Removendo uma tarefa
        gerenciador.removerTarefa(2);

        // Listando as tarefas novamente
        gerenciador.listarTarefas();
    }
}
```

Explicação do Código:

Criamos a classe GerenciadorDeTarefas com um **atributo tarefas**, que é uma ArrayList<String>.

Construtor da classe

- Inicializa a ArrayList para armazenar as tarefas.

Método adicionarTarefa(String descricao)

- Adiciona a tarefa à lista.
- Exibe uma mensagem confirmando a adição.

Método listarTarefas()

- Se a lista estiver vazia, exibe "Nenhuma tarefa cadastrada.".
- Caso contrário, percorre a lista e imprime cada tarefa numerada.

Método removerTarefa(int indice)

- Verifica se o índice fornecido é válido antes de remover.
- Remove a tarefa correspondente e exibe uma mensagem de confirmação.
- Se o índice for inválido, exibe "Índice inválido! Tente novamente.".

Método main()

- Cria uma instância de GerenciadorDeTarefas.
- Adiciona três tarefas.
- Lista as tarefas.
- Remove a segunda tarefa.
- Lista as tarefas novamente para verificar a remoção.

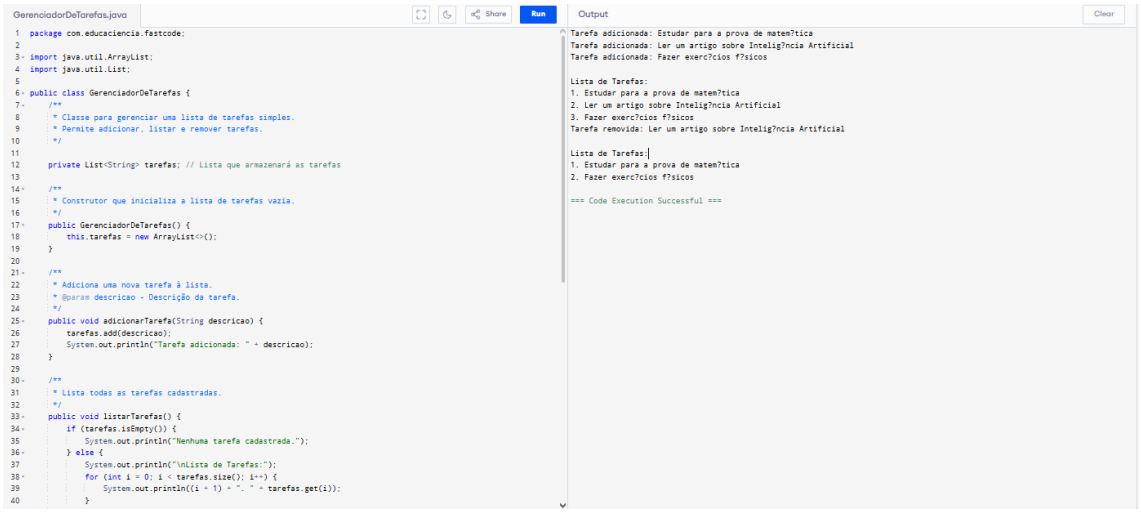
Saída esperada no console:

Tarefa adicionada: Estudar para a prova de matemática
Tarefa adicionada: Ler um artigo sobre Inteligência Artificial
Tarefa adicionada: Fazer exercícios físicos

Lista de Tarefas:
1. Estudar para a prova de matemática
2. Ler um artigo sobre Inteligência Artificial
3. Fazer exercícios físicos

Tarefa removida: Ler um artigo sobre Inteligência Artificial

Lista de Tarefas:
1. Estudar para a prova de matemática
2. Fazer exercícios físicos



Diferenças entre python e java nesse código

Funcionalidade	Python	Java
Estrutura da classe	class GerenciadorDeTarefas:	public class GerenciadorDeTarefas

Funcionalidade	Python	Java
Lista de tarefas	self.tarefas = []	private List<String> tarefas;
Construtor	def __init__(self):	public GerenciadorDeTarefas() {}
Adicionar tarefa	self.tarefas.append(descricao)	tarefas.add(descricao);
Listar tarefas	for i, t in enumerate(self.tarefas)	for (int i = 0; i < tarefas.size(); i++)
Remover tarefa	self.tarefas.pop(indice - 1)	tarefas.remove(indice - 1);
Verificação de índice	if 1 <= indice <= len(self.tarefas):	if (indice >= 1 && indice <= tarefas.size())
Método main	if __name__ == "__main__":	public static void main(String[] args) {}

Abraços e bons estudos !