

Variáveis, tipos e entrada do usuário

quarta-feira, 12 de novembro de 2025 09:57

- **Variável:** nome que referencia um valor em memória.
nome = "Ana" idade = 21
- **Tipos básicos:**
int (inteiro), float (real), str (texto), bool (True/False), NoneType (None).
- **Ver tipo:** type(x)
- **Conversão (casting):** int("10"), float("3.14"), str(42), bool(0)
- **Entrada:** input("Pergunta: ") → sempre retorna str
Ex.: idade = int(input("Idade: "))
- **Nomeação recomendada:** snake_case, nomes descritivos.
- **Atribuições úteis:**
múltipla: a, b = 1, 2 • *swap*: a, b = b, a • *incremento*: x += 1
- **f-string:** interpolação de variáveis com {}.
nome, n = "Ana", 7
print(f"Olá, {nome}. Você tem {n} mensagens.")
- **Formatação numérica:**
print(f"{3.14159:.2f}") # 3.14
print(f"{1234:08d}") # 00001234
print(f"{0.85:.0%}") # 85%
- **Expressões dentro de {}:** print(f"Dobro: {n*2}")
- **Formatação de alinhamento:**
print(f"{'Item':<10}{Qtd:>5}") (esq./dir.)

Operadores aritméticos

quarta-feira, 12 de novembro de 2025 14:40

- **Básicos:** + - * /
- **Divisão inteira:** // (descarta decimais) → $7//3 == 2$
- **Módulo (resto):** % → $7%3 == 1$
- **Potência:** ** → $2**3 == 8$
- **Precedência:** () > ** > * / // % > + -
- **Funções úteis:** abs(-5), round(3.456, 2)
Com math: import math → math.sqrt(9), math.pi, math.ceil(), math.floor()

Operadores lógicos e relacionais

quarta-feira, 12 de novembro de 2025 14:46

- **Relacionais:** == | != | > | >= | < | <=
3 == 3 → True
- **Lógicos:** and or not
maior_de_idade and tem_documento
- **Pertencimento:** in not in
"a" in "casa" # True
- **Identidade:** is not (mesmo objeto em memória)
- **Curto-circuito:** A and B só avalia B se A for True; A or B só avalia B se A for False.

Condições (if, elif, else)

quarta-feira, 12 de novembro de 2025 14:47

```
nota = float(input("Nota: "))
if nota >= 7:
    print("Aprovado")
elif nota >= 5:
    print("Recuperação")
else:
    print("Reprovado")
```

-
- **Condicional em uma linha (ternário):**
status = "par" if n % 2 == 0 else "ímpar"

- **match (3.10+):**
match opcao:
 case 1: print("Iniciar")
 case 2: print("Parar")
 case _: print("Inválido")

Repetições com for e while

quarta-feira, 12 de novembro de 2025 14:49

- **for com range:**

```
for i in range(5): ... # 0..4  
for i in range(1, 11, 2): ... # 1,3,5,7,9
```

- **Iterar coleções:**

```
for x in [10, 20, 30]: ...
```

- **Com índice:** for i, v in enumerate(lista): ...

- **Somar/filtrar** (compreensão de listas):

```
[x*x for x in nums if x%2==0]
```

- **while:**

```
while tentativa < 3 and not sucesso:
```

```
    tentativa += 1
```

- **Controle de laço:** break (sai), continue (pula), else do laço (executa se não houve break).

Listas, dicionários e tuplas

quarta-feira, 12 de novembro de 2025 14:51

Listas são **coleções ordenadas e mutáveis** (podem ser alteradas). Aceitam **tipos mistos** e são indexadas a partir de 0.

Criação

```
lista = [1, 2, 3, 4]
lista2 = ["Ana", "João", "Maria"]
lista_mista = [1, "texto", 3.5, True]
Também podem ser criadas com list():
numeros = list(range(5)) # [0, 1, 2, 3, 4]
```

Acesso e Fatiamento

```
lista[0] # primeiro elemento
lista[-1] # último elemento
lista[1:4] # do índice 1 ao 3
lista[:3] # primeiros 3
lista[2:] # do índice 2 em diante
lista[::-1] # lista invertida
```

Modificação de Elementos

```
lista[2] = 99 # altera o valor do índice 2
lista.append(10) # adiciona no final
lista.insert(1, 50) # insere no índice 1
lista.extend([7, 8]) # adiciona vários valores
```

Remoção de Elementos

```
lista.remove(50) # remove o valor 50 (primeira ocorrência)
del lista[0] # remove o índice 0
valor = lista.pop() # remove o último e retorna
valor = lista.pop(2) # remove o índice 2 e retorna
lista.clear() # limpa toda a lista
```

Métodos e Funções Úteis

Função / Método	Descrição	Exemplo
len(lista)	retorna o tamanho	len([1,2,3]) → 3
sum(lista)	soma elementos numéricos	sum([1,2,3]) → 6
max(lista) / min(lista)	maior / menor valor	max([1,4,2]) → 4
lista.count(x)	quantas vezes x aparece	[1,2,2,3].count(2) → 2
lista.index(x)	posição da primeira ocorrência	[1,2,3].index(2) → 1
lista.sort()	ordena em ordem crescente	[3,1,2].sort()
lista.sort(reverse=True)	ordem decrescente	[3,2,1]
sorted(lista)	retorna nova lista ordenada	sorted([3,1,2]) → [1,2,3]
lista.reverse()	inverte a ordem in place	[1,2,3] → [3,2,1]
reversed(lista)	retorna iterador invertido	list(reversed([1,2]))
copy()	cria uma cópia rasa	nova = lista.copy()

Diferença entre Cópia e Referência

```
a = [1, 2, 3]
b = a # MESMA lista na memória
c = a.copy() # NOVA lista (cópia independente)
Alterar b também muda a, mas alterar c não muda a.
```

Testes e Pertencimento

```
if 3 in lista:
    print("3 está na lista")
if 10 not in lista:
    print("10 não está na lista")
```

Iterações

```
for elemento in lista:
    print(elemento)
for i, valor in enumerate(lista):
    print(f"Índice {i}: {valor}")
```

Listas Aninhadas (listas dentro de listas)

```
matriz = [
    [1, 2, 3],
    [4, 5, 6]
]
print(matriz[0][1]) # 2
Muito usadas para representar matrizes, grades ou tabelas.
```

Compreensões de Lista (List Comprehension)

```
quadrados = [x**2 for x in range(6)] # [0,1,4,9,16,25]
pares = [x for x in range(10) if x % 2 == 0] # [0,2,4,6,8]
nomes = ["ana", "joao", "bia"]
maiúsculos = [n.upper() for n in nomes] # ['ANA','JOAO','BIA']
```

Com condição ternária:

```
resultado = ["Par" if x % 2 == 0 else "ímpar" for x in range(5)]
```

Aninhadas (duas dimensões):

```
pares_ate_3 = [(x, y) for x in range(3) for y in range(3) if x != y]
```

TUPLAS

- Imutáveis, ordenadas e indexadas.

```
t = (1, 2, 3)
print(t[0]) # 1
x, y, z = t # desempacotamento
# Usadas quando os dados não devem ser alterados.
# Convertendo: list(tupla) ou tuple(lista).
```

DICIONÁRIOS

Coleções não ordenadas de pares **chave → valor**.

```
pessoa = {"nome": "Ana", "idade": 25, "cidade": "Itajaí"}
print(pessoa["nome"])
pessoa["idade"] = 26
```

Métodos principais:

```
pessoa.keys() # todas as chaves
pessoa.values() # todos os valores
pessoa.items() # tuplas (chave, valor)
pessoa.get("email", "sem e-mail")
pessoa.update({"país": "Brasil"})
pessoa.pop("cidade")
```

Compreensão de dicionário:

```
quadrados = {x: x**2 for x in range(5)}
```

Uso com Funções Integradas

```
# Filtrar
lista_filtrada = list(filter(lambda x: x > 10, [5,12,8,20]))
```

```
# Transformar
lista_dobrada = list(map(lambda x: x*2, [1,2,3]))
```

Desempacotamento

```
valores = [10, 20, 30]
a, b, c = valores # a=10, b=20, c=30
# Desempacotar com resto
a, *meio, b = [1,2,3,4,5] # a=1, meio=[2,3,4], b=5
```

Cópias profundas (listas dentro de listas)

Quando há listas aninhadas, `.copy()` não é suficiente — use `copy.deepcopy()`.

```
import copy
matriz2 = copy.deepcopy(matriz)
```

Criando funções reutilizáveis

quarta-feira, 12 de novembro de 2025 15:00

```
def media(a, b=0, c=0):
    """Calcula média simples de até três números."""
    return (a + b + c) / 3
```

- **Parâmetros:** posicional, nomeado, *default*, **args* (variável), ***kwargs* (nomeados).
def soma(*nums): return sum(nums)
def config(**op): print(op.get("debug", False))
- **Docstring:** primeira string no corpo → help(func)
- **Escopo:** variáveis dentro da função são locais; use global/nonlocal com cuidado.
- **Funções anônimas:** lambda x: x*2 (curtas).
- **Boas práticas:** pequenas, coesas, nome claro, retorno explícito, sem efeitos colaterais desnecessários.

Modularização

quarta-feira, 12 de novembro de 2025 15:03

- **Estrutura:**

```
meu_projeto/
└── main.py
└── util.py
```

- **Em util.py:**

```
def saudacao(nome: str) -> str:
    return f"Olá, {nome}!"
```

- **Em main.py:**

```
import util      # importa o módulo inteiro
print(util.saudacao("Ana"))
from util import saudacao # importa só o símbolo
print(saudacao("Bob"))
```

- **Ponto de entrada:**

```
if __name__ == "__main__":
    # só executa quando rodar: python main.py
```

```
..."
```

- **Imports relativos em pacotes** (quando usar pastas com `__init__.py`):
from .util import saudacao
- **Executar módulo como script:** python -m pacote.submodulo

POO

quinta-feira, 13 de novembro de 2025 16:13

1) Classe básica e objeto

```
class Pessoa:  
    # atributo de classe (compartilhado)  
    especie = "Humano"  
  
    # construtor  
    def __init__(self, nome, idade):  
        self.nome = nome # atributo de instância  
        self.idade = idade  
  
    # método de instância  
    def apresentar(self):  
        print(f"Olá, meu nome é {self.nome} e tenho {self.idade} anos.")  
  
    # criando objetos  
p1 = Pessoa("Ana", 20)  
p2 = Pessoa("João", 25)  
  
p1.apresentar()  
p2.apresentar()  
print(Pessoa.especie)
```

4) Métodos de instância, de classe e estáticos

```
class Exemplo:  
    contador = 0 # atributo de classe  
  
    def __init__(self):  
        Exemplo.contador += 1  
  
    # método de instância (acessa self)  
    def metodo_instancia(self):  
        print("Instância:", self)  
  
    # método de classe (acessa a classe, não a instância)  
    @classmethod  
    def metodo_classe(cls):  
        print("Classe:", cls, "| contador:", cls.contador)  
  
    # método estático (função solta dentro da classe)  
    @staticmethod  
    def metodo_estatico(x, y):  
        return x + y  
  
e1 = Exemplo()  
e2 = Exemplo()  
e1.metodo_instancia()  
Exemplo.metodo_classe()  
print(Exemplo.metodo_estatico(2, 3))
```

2) Encapsulamento (público, "protegido", "privado")

```
class Conta:  
    def __init__(self, numero, saldo_inicial=0):  
        self.numero = numero # público  
        self._saldo = saldo_inicial # "protegido" (convenção)  
        self.__token = "secreto" # "privado" (name mangling)  
  
    def depositar(self, valor):  
        self._saldo += valor  
  
    def ver_saldo(self):  
        return self._saldo  
  
c = Conta("123-4", 100)  
c.depositar(50)  
print(c.ver_saldo()) # OK  
print(c.numero) # OK  
# print(c._saldo) # possível, mas NÃO recomendado  
# print(c.__token) # ERRO  
# print(c._Conta__token) # funciona (name mangling), mas é gambiarra
```

3) @property (getters e setters "bonitos")

```
class Produto:  
    def __init__(self, nome, preco):  
        self.nome = nome  
        self._preco = preco  
  
    @property  
    def preco(self):  
        return self._preco  
  
    @preco.setter  
    def preco(self, valor):  
        if valor < 0:  
            raise ValueError("Preço não pode ser negativo")  
        self._preco = valor  
  
p = Produto("Mouse", 50)  
print(p.preco) # chama getter  
p.preco = 80 # chama setter  
# p.preco = -10 # levanta erro
```

5) Herança simples

```
class Animal:  
    def __init__(self, nome):  
        self.nome = nome  
  
    def emitir_som(self):  
        print("Som genérico...")  
  
class Cachorro(Animal): # herda de Animal  
    def emitir_som(self):  
        print(f"{self.nome} diz: Au au!")  
  
a = Animal("Bicho")  
c = Cachorro("Rex")  
  
a.emitir_som()  
c.emitir_som() # polimorfismo (mesmo nome, comportamento diferente)
```

6) super() (chamar método da classe pai)

```
class Funcionario:  
    def __init__(self, nome, salario):  
        self.nome = nome  
        self.salario = salario  
  
    def info(self):  
        return f"Funcionário: {self.nome} - Salário: {self.salario}"  
  
class Gerente(Funcionario):  
    def __init__(self, nome, salario, setor):  
        # chama o construtor da classe pai  
        super().__init__(nome, salario)  
        self.setor = setor  
  
    def info(self):  
        base = super().info()  
        return base + f" - Setor: {self.setor}"  
  
g = Gerente("Carla", 5000, "TI")  
print(g.info())
```

7) Herança múltipla

```
class Logavel:  
    def logar(self, msg):  
        print(f"[LOG] {msg}")  
  
class Autenticavel:  
    def autenticar(self, senha):  
        return senha == "1234"  
  
class Usuario(Logavel, Autenticavel):  
    def __init__(self, nome):  
        self.nome = nome  
  
u = Usuario("Henrique")  
u.logar("Usuário criado")  
print("Autenticado.", u.autenticar("1234"))
```

8) Herança multinível (vários "degraus")

```
class SerVivo:  
    def respirar(self):  
        print("Respirando...")  
  
class Animal(SerVivo):  
    def mover(self):  
        print("Movendo-se...")  
  
class Cachorro(Animal):  
    def latir(self):  
        print("Au au!")  
  
dog = Cachorro()  
dog.respirar() # herdado de SerVivo  
dog.mover() # herdado de Animal  
dog.latir() # da própria classe
```

9) Polimorfismo (mesma interface, classes diferentes)

```
class Forma:  
    def area(self):  
        raise NotImplementedError  
  
class Quadrado(Forma):  
    def __init__(self, lado):  
        self.lado = lado  
  
    def area(self):  
        return self.lado ** 2  
  
class Retangulo(Forma):  
    def __init__(self, largura, altura):  
        self.largura = largura  
        self.altura = altura  
  
    def area(self):  
        return self.largura * self.altura
```

```
formas = [Quadrado(2), Retangulo(2, 5)]  
for f in formas:  
    print("Área:", f.area()) # chama o método correto pra cada tipo
```

10) Classe abstrata (ABC – Abstract Base Class)

```
from abc import ABC, abstractmethod  
class Veiculo(ABC):  
    @abstractmethod  
    def ligar(self):  
        pass
```

11) Composição (usar um objeto dentro do outro)

Não é herança, mas é muito usado em POO:

```
class Motor:  
    def ligar(self):  
        print("Motor ligado.")
```

```
from abc import ABC, abstractmethod
class Veiculo(ABC):
    @abstractmethod
    def ligar(self):
        pass
    @abstractmethod
    def desligar(self):
        pass
class Carro(Veiculo):
    def ligar(self):
        print("Carro ligado.")
    def desligar(self):
        print("Carro desligado.")
# v = Veiculo() # ERRO: não pode instanciar
classe abstrata
c = Carro()
c.ligar()
c.desligar()
```

Não é herança, mas é muito usado em POO:

```
class Motor:
    def ligar(self):
        print("Motor ligado.")
    def desligar(self):
        print("Motor desligado.")
class Carro:
    def __init__(self):
        self.motor = Motor() # Carro TEM UM motor (composição)
    def ligar(self):
        self.motor.ligar()
    def desligar(self):
        self.motor.desligar()
carro = Carro()
carro.ligar()
carro.desligar()
```

Estrutura de dados

terça-feira, 6 de janeiro de 2026 17:42

1. Estrutura Básica do HTML

- <!DOCTYPE html> – Define o documento como HTML5.
- <html> – Raiz do documento.
- <head> – Metadados (título, charset, CSS, scripts, SEO).
- <body> – Conteúdo visível da página.

2. Cabeçalhos e Títulos

- <h1> até <h6> – Hierarquia de títulos, sendo <h1> o mais importante.

3. Parágrafos

- <p> – Parágrafo de texto.
-
 – Quebra de linha.
- <hr> – Linha horizontal divisória.

4. Formatação de Texto

- **Negrito:** (semântico) ou
- **Itálico:** (semântico) ou <i>
- **Sublinhado:** <u>
- **Marcador de texto:** <mark>
- **Pequeno:** <small>
- **Riscado:**
- **Inserido:** <ins>
- **Código:** <code>, <pre>

5. Listas

Ordenadas

-
 -

Não ordenadas

-
 -

Listas de descrição

- <dl>
 - <dt> – termo
 - <dd> – descrição

6. Imagens

-
- Atributos importantes:
 - width, height
 - title
 - loading="lazy"

7. Links

- texto
- Atributos úteis:
 - target="_blank" – abre em nova aba
 - rel="noopener" – segurança
 - download – baixa arquivo

8. Tabelas

- <table> – tabela
- <tr> – linha
- <th> – cabeçalho
- <td> – célula
- <caption> – legenda
- <thead>, <tbody>, <tfoot> – organização
- Atributos comuns: rowspan, colspan

9. Mais Recursos de Tabela

- Agrupamento visual
- Efeito zebra (via CSS)
- Células mescladas
- Cabeçalhos fixos (CSS/JS)

10. Formulários

- <form> – inicia um formulário
- Atributos: action, method="GET/POST"

Campos principais

- <input> – texto, número, email, senha, arquivo, cor, data etc.
- <textarea> – texto multilinha
- <label> – rótulo
- <button> – botão

11. Tipos comuns de <input>

- text, number, email, password, date,
- range, color, file, checkbox, radio,
- submit, reset, hidden, search

12. Mais Componentes de Formulários

- <select> + <option> – listas suspensas
- <optgroup> – agrupar opções
- <fieldset> – agrupar campos
- <legend> – título do fieldset
- Atributos: required, placeholder, maxlength, pattern, disabled, readonly, checked, selected

13. Caracteres Especiais (HTML Entities)

- &nbsp – espaço
- < <
- > >
- & &
- © ©
- ® ®
- € €
- " " "
- ' ''

❖ Conteúdo Extra Importante (Intermediário)

14. Divisões e Semântica

Div padrão:

- <div> – bloco genérico

Tags semânticas modernas (HTML5):

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <footer>
- <main>
- <figure> + <figcaption>

15. Elementos Inline Comuns

- – estilização inline
- <abbr> – abreviação
- <cite> – citação
- <q> – citação curta

16. Mídias

Áudio:

- <audio controls>
 - <source src="">

Vídeo:

- <video controls>
 - <source src="">

Iframe:

- <iframe src=""> – embutir sites, mapas, vídeos etc.

17. Meta Tags Importantes

- <meta charset="UTF-8">
- <meta name="viewport" content="width=device-width, initial-scale=1.0">
- SEO:
 - <meta name="description" content="">
 - <meta name="keywords" content="">

18. Comentários

- <!-- comentário -->

19. Scripts e Estilos

CSS

- <link rel="stylesheet" href="">
- <style> – CSS interno

JavaScript

- <script>
- <script src=""> – arquivo externo

20. Atributos Globais Importantes

Disponíveis em quase todas as tags:

- id – identificação única
- class – classes CSS
- style – estilo inline
- title – tooltip
- hidden
- lang
- data-* – atributos personalizados

➊ 21. Testando Conhecimentos — O que costuma ser cobrado

- Criar tabelas completas
- Formulários com validação básica
- Estruturação com tags semânticas
- Links com atributos
- Inserção de mídia
- Uso de listas dentro de listas
- Criar layout básico com <header>, <nav>, <main> etc.

1. Tipos de Inserção de CSS

Inline

O estilo é aplicado diretamente no elemento HTML usando o atributo style.

Usa: ajustes rápidos e pontuais.

Internal

O CSS é colocado dentro da tag <head> no <head>.

Usa: páginas pequenas ou testes locais.

External

O CSS fica em um arquivo externo .css e é importado com <link>.

Usa: projetos organizados, reutilização e manutenção melhor.

2. Seletores

Seletores Básicos

Elemento: seleciona todos elementos de um tipo (ex.: p).

Classe: seleciona elementos da mesma classe (ex.: .btn).

ID: seleciona um único elemento com ID específico (ex.: #menu).

Universal: seleciona todos os elementos (*).

Combinadores

Descendente: A seleciona elementos B dentro de A.

Filho: A > B seleciona elementos B diretamente dentro de A.

Adjacente: A + B seleciona B imediatamente após A.

General Irmãos: A ~ B seleciona B que compartilha o mesmo pai de A.

Seletores de Atributo

Permite selecionar elementos com base em atributos:

- Exato: [type="text"]
- Começa com: [href^="https"]
- Termina com: [href\$=".pdf"]
- Contém: [href*="#login"]

3. Cores

• color: cor do texto.

• font-color: nome da fonte.

• font-family: tipo da fonte.

• font-weight: peso (espessura).

• font-style: itálico, normal etc.

• text-align: alinhamento horizontal.

• text-transform: maiúscula/minúscula/capitalizar.

• text-decoration: sublinhado, linha, etc.

• line-height: altura da linha.

• letter-spacing: espaçamento entre letras.

5. Background (Plano de Fundo)

• background-color: cor de fundo.

• background-image: imagem de fundo.

• background-repeat: repetição (repeat/no-repeat).

• background-size: controle do tamanho da imagem (cover/contain).

• background-position: posição da imagem.

• background-attachment: fixa ou rola com a página.

6. Bordas

• border: cria borda (cor, estilo, espessura).

• border-width: espessura.

• border-style: sólido, pontilhado etc.

• border-color: cor.

• border-radius: arredondamento dos cantos.

7. Espaçamento

Margin

Espaço externo do elemento (fora da caixa).

Pode ser individual (top/right/bottom/left).

Padding

Espaço interno do elemento (entre o conteúdo e a borda).

Também pode ser individual.

8. Dimensões

- width/height: define largura e altura.
- max-width/max-height: limites máximos.
- min-width/min-height: limites mínimos.

9. Display e Layout

- block: ocupa toda linha, quebra linhas.
- inline: ocupa apenas o necessário, não quebra linha.
- inline-block: inline, porém aceita width/height.
- flex: ativa flexbox.
- grid: ativa grid.
- none: esconde o elemento.
- overflow: controla conteúdo que ultrapassa limites (scroll/hidden).
- visibility: mostra ou esconde sem alterar espaço.

10. Position

- static: padrão, sem posicionamento especial.
- relative: referência para deslocamentos.
- absolute: posicionado em relação ao ancestral posicionado.
- fixed: fixo na tela, mesmo ao rolar.
- sticky: alterna entre relative e fixed conforme rolagem.
- z-index: controla a ordem de sobreposição.

Conceito: layout unidimensional (linha ou coluna).

11. Flexbox (Resumo Teórico)

- display: flex ativa o flexbox.
- flex-direction: define direção (row/column).
- justify-content: alinha no eixo principal.
- align-items: alinha no eixo cruzado.
- flex-wrap: permite quebra de linha.
- gap: espaçamento entre itens.

Conceito: layout bidimensional (linha e coluna juntas).

12. Grid (Resumo Teórico)

- display: grid ativa grid.
- grid-template-columns: define colunas.
- grid-template-rows: define linhas.
- gap: espaçamento.
- grid-template-areas: cria áreas nomeadas.
- place-items: centraliza elementos.

Conceito: layout bidimensional (linha e coluna juntas).

13. Listas (Estilo de Listas)

- list-style-type: tipo de marcador (disc, square, none).
- list-style-position: posição do marcador (inside/outside).
- list-style-image: usa imagem como marcador.

14. Tabelas (Estilos)

- border-collapse: une bordas.
- border-spacing: espaçamento entre células.
- caption-side: local da legenda (top/bottom).

1. Quando usar Flexbox?

Use quando você quer:

- Distribuir elementos em linha (horizontal)
 - Distribuir elementos em coluna (vertical)
 - Centralizar itens facilmente
 - Criar layouts simples e flexíveis
 - Ajustar automaticamente conforme o espaço disponível
- Flexbox é ótimo para componentes, não para layouts completos.

2. Quando usar Grid?

Use quando você quer:

- Trabalhar com linhas + colunas ao mesmo tempo
 - Definir áreas completas do layout (header, menu, conteúdo, sidebar, footer)
 - Criar grades complexas
 - Criar grids responsivos sem precisar de media queries
 - Controlar o posicionamento pelos dois eixos simultaneamente
- Grid é ótimo para layout de página e organização avançada.

3. Diferença prática (resumo rápido)

Característica	Flexbox	Grid
Dimensão	1D (linha OU coluna)	2D (linha E coluna)
Melhor uso	Componentes	Layout completo
Direção	Um eixo por vez	Dois eixos ao mesmo tempo
Controle	Tamanho dos itens	Tamanho dos itens e do layout
Alinhamento	Muito fácil e flexível	Muito preciso e planejado
Responsividade	Excelente	Excelente, porém mais estruturada

15. Imagens

- object-fit: como a imagem se ajusta (cover/contain).
- object-position: posição dentro do container.

16. Cursos

- cursor: define tipo do cursor (pointer/text/move/etc.).

17. Transições (Transitions)

- transition-property: o que será animado.
- transition-duration: tempo da transição.
- transition-delay: atraso antes de começar.
- transition-timing-function: forma da transição (ease/linear).

Conceito: animação suave entre dois estados.

18. Transformações (Transform)

- scale: aumenta/diminui.
- rotate: gira.
- translate: move o elemento.
- skew: inclina.

19. Animações

- @keyframes: define a animação.
- animation-name: nome da animação.
- animation-duration: tempo total.
- animation-iteration-count: quantas vezes repete.
- animation-delay: atraso antes de iniciar.

20. Sombras

- box-shadow: sombra de caixas.
- text-shadow: sombra de texto.

21. Opacidade

- opacity: controla transparência (0 = invisível, 1 = opaco).

22. Outros Componentes

filter

Aplica efeitos como blur, brilho, saturação.

backdrop-filter

Aplica efeitos atrás do elemento (vídeo fosco).

pointer-events

Controla se o elemento pode ser clicado.

user-select

Controla se o usuário pode selecionar o texto.

ID (Identificador Único)

O que é:

É um identificador exclusivo para um único elemento na página.

Características:

- Deve existir apenas 1 vez por página.
- Usado quando você precisa identificar um elemento específico.
- Usado no HTML com id="algumNome".
- Selecionado no CSS com #algumNome.

Quando usar ID?

- Para identificar elementos únicos, como:
 - Header principal
 - Rodapé
 - Banner
 - Seções exclusivas
 - Elemento que será manipulado por JavaScript

Exemplo HTML:

```
<h1 id="titulo-principal">Bem-vindo</h1>
```

Exemplo CSS:

```
#titulo-principal {  
  color: blue;  
}
```

Classe (class)

O que é:

Uma marcação reutilizável que pode ser aplicada a vários elementos.

Características:

- Pode ser usada quantas vezes quiser.
- Um elemento pode ter várias classes ao mesmo tempo.
- Usado no HTML como class="algumaClasse".
- Selecionado no CSS com .algumaClasse.

Quando usar classe?

- Quando vários elementos têm o mesmo estilo.
- Para botões, cartões, listas, cabeçalhos, layouts repetidos etc.

Exemplo HTML:

```
<p class="texto-destaque">Olá!</p>
```

```
<p class="texto-destaque">Mundo!</p>
```

Exemplo CSS:

```
.texto-destaque {  
  font-weight: bold;  
  color: red;  
}
```

Diferença resumida (ideal para prova)

Recurso	ID	Classe
Quantidade por página	Apenas 1	Ilimitado
Reutilização	Não	Sim
CSS	#id	.classe
Ideal para	Elementos únicos	Vários elementos com o mesmo estilo

Explicação em uma frase

- ID = único.
- Classe = reutilizável.

1. Seletores Avançados

Seletores de Atributo

Permitem estilizar elementos com base nos atributos HTML.

- [type="text"] – valor exato
- [href^="https://"] – começa com
- [href\$=".pdf"] – termina com
- [href*="login"] – contém

Para que serve?

Criar estilos específicos sem precisar adicionar classes em tudo.

Seletores de Irmãos e Descendência

- A B – descendente (qualquer nível interno)
- A > B – filho direto
- A + B – irmão imediatamente seguinte
- A ~ B – irmãos gerais

Para que serve?

Criar dependência de estilo com base na posição dos elementos.

Pseudo-classes

São estados especiais do elemento.

- :hover – quando o mouse passa
- :active – quando clicado
- :focus – quando selecionado (inputs)
- :first-child – primeiro filho
- :last-child – último filho
- :nth-child(n) – posição específica
- :not(X) – tudo menos X

Para que serve?

Criar comportamentos interativos e seleções inteligentes.

Pseudo-elementos

Criam partes virtuais do elemento.

- ::before
- ::after
- ::first-letter
- ::first-line

Para que serve?

Criar conteúdo adicional ou estilizar partes específicas de texto.

2. Herança

Herança é o processo em que algumas propriedades são passadas automaticamente dos elementos-pai para os elementos-filho.

Propriedades que herdam:

- color
- font-family
- font-size
- line-height

Propriedades que não herdam:

- border
- margin
- padding
- width/height

Para que serve?

Evitar repetição e tornar o CSS mais limpo.

3. Especificidade

É a regra que define quem vence no CSS quando dois estilos conflitam.

Ordem de força (fraco → forte):

1. Seletor universal (*)
2. Elemento (p, h1)
3. Classe, pseudo-classe (.btn, :hover)
4. ID (#menu)
5. Inline (style="...")
6. !important (evita ao máximo)

Para que serve?

Diagnosticar problemas quando o estilo "não funciona".

4. Box Model Avançado

Todo elemento é uma caixa composta de:

1. Content – texto/elementos internos
 2. Padding – espaço interno
 3. Border – borda
 4. Margin – espaço externo
- E também:
- box-sizing: content-box; (padrão)
 - box-sizing: border-box; (recomendado)
- (Faz width/height incluirem border e padding)

Para que serve?

Controlar layout com precisão.

5. Display (intermediário/avançado)

- block – ocupa a linha toda
- inline – ocupa só o conteúdo
- inline-block – inline + aceita tamanho
- none – remove elemento
- flex – ativa o Flexbox
- grid – ativa o Grid
- inline-flex / inline-grid – versões internas

Para que serve?

Definir como o elemento se comporta no fluxo da página.

6. Position (avançado)

- static – padrão
- relative – serve como referência
- absolute – sai do fluxo, usa ancestral posicionado
- fixed – preso na tela
- sticky – gruda até certo ponto
- z-index – define quem fica por cima

Para que serve?

Criar pop-ups, barras fixas, overlays, banners.

7. Flexbox (Essência Avançada)

Flexbox controla um eixo por vez (horizontal ou vertical):

- display: flex
- flex-direction: row | column
- justify-content: eixo principal
- align-items: eixo cruzado
- flex-wrap – permite quebrar linha
- flex-grow – item crescer
- flex-shrink – item encolher
- flex-basis – tamanho base
- gap – espaçamento

Para que serve?

Componentes, menus, cartões, caixas responsivas.

8. Grid (Essência Avançada)

Grid controla duas dimensões ao mesmo tempo (linha + coluna):

- display: grid
- grid-template-columns
- grid-template-rows
- grid-template-areas
- gap
- place-items

Unidades importantes:

- fr – fração do espaço disponível
- minmax() – mínimo e máximo
- repeat() – repetição inteligente

Para que serve?

Layouts completos, painéis, dashboards, galerias.

9. Layouts Fixos e Líquidos

Layout Fijo

Usa valores absolutos (px).

Não se adapta ao tamanho da tela.

Layout Líquido

Usa %, vh, vw para adaptar ao espaço.

Para que serve?

Fazer layouts抗igos (fixos) ou modernos (responsivos).

10. Imagens Líquidas

- max-width: 100%;
- height: auto;

Para que serve?

Fazer imagens reduzirem em telas menores sem quebrar layout.

11. Coluna Falsa (Fake Column)

Técnica antiga para criar layout com colunas de altura igual antes da existência do Flexbox e Grid.

Hoje:

Basta usar:

- display: flex; align-items: stretch;
- ou
- display: grid;

12. Navegação (menus)

Barra Vertical

- Lista () organizada em coluna
- Links como blocos inteiros (display: block)

Barra Horizontal

- Itens em linha
- Pode usar inline-block ou flex

13. Marcando página atual

Usa-se normalmente uma classe:

```
<a class="ativo">Home</a>
```

E o CSS:

```
.ativo {
  background-color: #ccc;
  font-weight: bold;
}
```

Para que serve?

Destacar em qual página o usuário se encontra.

14. Estilizando Tabelas

- border-collapse
- Cores alternadas (nth-child)
- Linhas zebra
- Cabeçalho destacado
- Hover nas linhas

Para que serve?

Deixar tabelas legíveis e modernas.

15. Estilizando Formulários

- Estilos em <input> e <textarea>
- Estados :focus
- Bordas, padding
- Botões estilizados
- Agrupamentos com <fieldset>

Para que serve?

Melhorar a aparência e a usabilidade.

16. Parallax (avanço gráfico)

O fundo se move mais devagar que o conteúdo.

Conceito básico:

- background-attachment: fixed;
- Imagens grandes
- Seções altas

Para que serve?

Criar efeito visual de profundidade.

17. Fontes Customizadas

- @font-face importa fontes do computador
- link importa do Google Fonts
- Pode usar qualquer fonte externa

Para que serve?

Personalizar tipografia e identidade visual do projeto.

Como usar (Importação via CDN)

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
```

Estrutura:

container > row > col

Tipos de Container:

- .container → largura fixa por breakpoint
- .container-fluid → 100% da largura
- .container-{breakpoint} → controla quando fixa

Breakpoints:

Prefixo Tamanho

col-	0-576px
col-sm-	≥576px
col-md-	≥768px
col-lg-	≥992px
col-xl-	≥1200px
col-xxl-	≥1400px

Exemplo de layout:

```
<div class="row">  
  <div class="col-12 col-md-6 col-lg-4">Coluna</div>  
</div>
```

Classes Utilitárias

✓ Espaçamento

- m-0 a m-5 → margin
- p-0 a p-5 → padding
- mt-3, mb-4, px-2, etc.

✓ Flexbox

- d-flex
- justify-content-center
- align-items-center
- flex-column
- gap-3

✓ Cores

- Texto: text-primary, text-warning
- Fundo: bg-dark, bg-success
- Bordas: border, border-danger

✓ Display

- d-none
- d-block
- d-md-flex (por breakpoint)

✓ Botões

```
<button class="btn btn-primary">Enviar</button>  
<button class="btn btn-outline-danger">Cancelar</button>
```

✓ Cards

```
<div class="card" style="width:18rem">  
    
  <div class="card-body">  
    <h5 class="card-title">Título...</h5>  
    <p class="card-text">Texto...</p>  
  </div>  
</div>
```

✓ Navbar

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">  
  <div class="container-fluid">  
    <a class="navbar-brand">Logo</a>  
    <button class="navbar-toggler" data-bs-toggle="collapse" data-bs-target="#menu">  
      <span class="navbar-toggler-icon"></span>  
    </button>  
    <div id="menu" class="collapse navbar-collapse">  
      <ul class="navbar-nav ms-auto">  
        <li class="nav-item active"><a class="nav-link active" href="#">Home</a></li>  
      </ul>  
    </div>  
  </div>  
</nav>
```

✓ Modal

```
<button class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#modal">  
  Abrir</button>  
<div class="modal fade" id="modal">  
  <div class="modal-dialog">  
    <div class="modal-content">...</div>  
  </div>  
</div>
```

✓ Carousel (Slides)

Ideal para imagens e destaque visual.

Formulários com Bootstrap

Input

```
<input class="form-control" placeholder="Digite seu nome">  
Select
```

```
<select class="form-select">  
  <option>Opção 1</option>  
</select>
```

Checkbox/Radio

```
<div class="form-check">  
  <input class="form-check-input" type="checkbox">  
  <label class="form-check-label">Aceito os termos</label>  
</div>
```

Bootstrap Icons

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css">  
<i class="bi bi-alarm"></i>
```

Customização (Override)

```
.btn-primary {  
  background-color: #6d00ff !important;  
}
```

2. Conceitos Básicos

2.1. Variáveis

```
var nome = "Ana"; // escopo global
let idade = 20; // escopo de bloco
const PI = 3.14; // constante
```

2.2. Tipos Primitivos

- string
- number
- boolean
- undefined
- null
- bigint
- symbol

2.3. Operadores

Aritméticos

```
+ - * / % **
```

Comparação

```
== === != != > < >= <=
```

Lógicos

```
&& || !
```

Atribuição

```
= += -= *= /= %=
```

Ternário

```
let status = idade >= 18 ? "adulto" : "menor";
```

3. Estruturas de Controle

3.1. Condicionais

```
if (condição) {
} else if (condição2) {
} else {
}
```

switch

```
switch (dia) {
  case 1:
    break;
  default:
    break;
}
```

3.2. Loops

```
for (let i = 0; i < 10; i++) {}
while (condição) {
  do {} while (condição);
  for...of (iterável)
}
```

```
for (let item of lista) {}
for...in (propriedades)
```

```
for (let chave in objeto) {}
```

4. Funções

4.1. Função padrão

```
function soma(a, b) {
  return a + b;
}
```

4.2. Função anônima

```
const soma = function(a, b) {
  return a + b;
}
```

4.3. Arrow Function

```
const soma = (a, b) => a + b;
```

4.4. Parâmetros padrão

```
function mult(a = 1, b = 1) {
  return a * b;
}
```

5. Arrays

Criação

```
let arr = [1, 2, 3];
```

Métodos importantes

Modificam o array

- push()
- pop()
- shift()
- unshift()
- splice()

Não modificam o array

- slice()
- map()
- filter()
- reduce()
- find()
- includes()

Exemplo:

```
arr.map(x => x * 2);
```

6. Objetos

Exemplo:

```
let pessoa = {
  nome: "Ana",
  idade: 20,
  falar() {
    console.log("Olá");
  }
};
```

Acesso:

```
pessoa.nome;
pessoa["idade"];
```

7. JSON

Formato usado para troca de dados.

```
JSON.stringify(obj); // objeto -> string
JSON.parse(texto); // string -> objeto
```

8. DOM (Document Object Model)

Seleção de elementos

```
document.getElementById("id");
document.querySelector(".classe");
document.querySelectorAll("p");
```

Alterando conteúdo

```
element.innerHTML = "texto";
element.textContent = "texto puro";
```

Estilos

```
element.style.color = "red";
```

Eventos

```
element.addEventListener("click", () => {
  console.log("clicado");
});
```

9. Eventos Comuns

- click
- mouseover
- keydown
- submit
- input
- change

Exemplo:

```
document.addEventListener("keydown", e => {
  console.log(e.key);
});
```

10. Assíncrono (Assynchronous)

10.1. Callbacks

```
setTimeout(() => {
  console.log("executou");
}, 1000);
```

10.2. Promises

```
let p = new Promise((resolve, reject) => {
  resolve("OK");
});
p.then(res => console.log(res));
```

10.3. async/await

```
async function carregar() {
  let resposta = await fetch(url);
}
```

11. Fetch API

```
fetch("https://api.com")
  .then(r => r.json())
  .then(data => console.log(data))
  .catch(e => console.log(e));
Com async/await:
```

```
async function exemplo() {
  const r = await fetch(url);
  const dados = await r.json();
}
```

12. Classes (ES6)

```
class Pessoa {
  constructor(nome, idade) {
    this.nome = nome;
    this.idade = idade;
  }
  falar() {
    console.log("Olá");
  }
}
let p = new Pessoa("Ana", 20);
```

13. Módulos

Exportar:

```
export function soma() {}
export default MinhaClasse;
```

Importar:

```
import { soma } from "./utils.js";
import MinhaClasse from "./Classe.js";
```

14. Tratamento de Erros

```
try {
  // código
} catch (erro) {
  console.log(erro.message);
} finally {
  // sempre executa
}
```

15. BOM (Browser Object Model)

Objetos do navegador:

- window
- history
- screen
- navigator
- location

Exemplo:

```
window.location.href;
```

- Servidores com http/express

16. LocalStorage e SessionStorage

Salvar

```
localStorage.setItem("nome", "Ana");
```

Ler

```
localStorage.getItem("nome");
```

Remover

```
localStorage.removeItem("nome");
```

17. Higher-Order Functions

Funções que recebem ou retornam outras funções.

Exemplos:

- map
- filter
- reduce

```
arr.filter(x => x > 10);
```

18. Desestruturação

Objetos

```
const { nome, idade } = pessoa;
```

Arrays

```
const [a, b] = [10, 20];
```

19. Spread e Rest

Spread

```
let arr2 = [...arr, 4, 5];
```

Rest

```
function soma(...nums) {
  return nums.reduce((t, n) => t + n);
}
```

20. Prototypes e Herança

```
function Pessoa(nome) {
  this.nome = nome;
}
```

```
Pessoa.prototype.falar = function() {
  console.log("Olá");
};
```

21. Async Patterns Importantes

- async/await
- promises em cadeia
- Promise.all
- Promise.race

22. Node.js (Resumo rápido)

- Não tem DOM
- Usa require/import
- Acesso ao sistema de arquivos (fs)

REACT

segunda-feira, 24 de novembro de 2025 12:10

Fazer upload dos projetos

quarta-feira, 12 de novembro de 2025 11:15

Caso já tenha um projeto pronto

Criar repositório no GitHub:

Crie um novo repositório no site do GitHub clicando em **New Repository**.
Exemplo: nome do repositório = MeuRepositorio.

1. Criar uma pasta no local que você vai deixar o projeto
`mkdir "C:\Users\name\Documents\nome_projeto"`
2. Entrar na pasta criada
`cd "C:\Users\name\Documents\nome_projeto"`
3. Inicializar o repositório Git local
`git init`
4. Criar uma pasta do projeto dentro do repositório
`mkdir "projeto1"`
5. Criar um arquivo de exemplo dentro da pasta
`echo "print('Hello World')" > projeto1\main.py`
6. Adicionar os arquivos ao controle de versão
`git add .`
7. Criar o primeiro commit
`git commit -m "Adicionado projeto1"`
8. Conectar o repositório local ao repositório remoto do GitHub
`git branch -M main`
`git remote add origin https://github.com/SEU_USUARIO/MeuRepositorio.git`
(esse aparece logo quando você cria um repositório no github)
9. Enviar os arquivos para o GitHub
`git push -u origin main`
10. Adicionar outro projeto dentro do mesmo repositório
`mkdir "projeto2" echo "print('Segundo projeto')" > projeto2\app.py`
`git add .`
`git commit -m "Adicionado projeto2"`
`git push origin main`

1. Criar repositório no GitHub
Crie um novo repositório no site do GitHub clicando em **New Repository**.
Exemplo: nome do repositório = MeuRepositorio.

2. Clonar o repositório para o computador
`git clone https://github.com/SEU_USUARIO/MeuRepositorio.git`

3. Entrar na pasta do repositório
`cd "C:\Users\name\Documents\MeuRepositorio"`

4. Criar uma pasta para o novo projeto dentro do repositório
`mkdir "projeto3"`

5. Mover o projeto já pronto para dentro dessa nova pasta

`move "C:\Users\name\Documents\meu_projeto_pronto" "C:\Users\name\Documents\MeuRepositorio\projeto3"`

6. Verificar se os arquivos foram movidos corretamente
`dir`

7. Adicionar todos os arquivos ao controle de versão
`git add .`

8. Criar o commit com uma mensagem descritiva
`git commit -m "Adicionado projeto3 (projeto pronto importado)"`

9. Enviar as alterações para o GitHub
`git push origin main`

R

segunda-feira, 19 de agosto de 2024 08:29

1. Configuração do Ecosistema e Ingestão

Para que serve: Iniciaiza as bibliotecas de processamento matricial (numpy), estruturação de dados (pandas) e visualização de dados (matplotlib, seaborn, plotly).

Por que é importante: Centraliza o controle de estilos e dependências, garantindo reprodutibilidade e performance no tratamento de grandes volumes de dados.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
# Carregando o dataset de renda (Tutorial)
df = pd.read_csv('arquivo.csv')
df.set_index('Variavel_Chave', inplace=True) # Indexar facilita consultas via .loc
```

2. Estruturação e Subsetting (Filtros)

Para que serve: Selecionar fatias específicas do dataset e converter estruturas de memória (como Series) em novos datasets.

Por que é importante: Permite isolat entidades (ex: um país ou uma classe social) para análise profunda sem interferência de ruidos externos do dataset global.

```
Python
```

```
# Exemplo Prático: Filtragem e Criação de DF
```

```
selecao = df.loc['Brazil', :].reset_index() # Extra linha e colunas temporais
```

```
dados_brasil = pd.DataFrame({'Ano': selecao.index, 'Valor': selecao.values})
```

3. Discretização e Cruzamento (Crosstab)

Para que serve: Transformar dados contínuos em categorias (pd.cut) e realizar contagens de frequência cruzadas entre variáveis (pd.crosstab).

Por que é importante: Essencial para entender correlações e distribuições entre categorias (ex: Renda vs Sexo ou Cor), revelando paixões sociodemográficas ocultas.

```
Python
```

```
# Tutorial: Binning e Crosstab
```

```
classes = [min, conte1, conte2, max]
```

```
labels = ['A', 'B', 'C', 'D']
```

```
df['Categoria'] = pd.cut(df['Varavel'], bins=classes, labels=labels)
```

```
# Tabela de contagem da Cruzada
```

```
frequencia = pd.crosstab(df['Varavel'], A, df['Varavel_B'])
```

```
Python
```

4. Medidas de Tendência Central e Assimetria

Para que serve: Calcular a Média (centro de massa), Mediana (valor central) e Moda (valor mais frequente).

Por que é importante: A relação entre essas medidas indica a Assimetria da distribuição. Se Média > Modiana, há uma assimetria à direita (cauda longa em valores altos), comum em dados de renda.

```
Python
```

```
media = df['Varavel'].mean() # Sensível a outliers
```

```
mediana = df['Varavel'].median() # Robusta a outliers
```

```
moda = df['Varavel'].mode() # Ponto de maior densidade
```

5. Medidas de Dispersão (Variabilidade)

Para que serve: Quantificar o quanto "espalhados" os dados estão em relação ao centro (Desvio Padrão e Variância).

Por que é importante: Define a confiabilidade da média. Um Desvio Padrão elevado indica que o dataset é heterogêneo, e que os dados fluem muito.

```
Python
```

```
desvio_padrão = df['Varavel'].std() # Raiz quadrada da variancia
```

```
variancia = df['Varavel'].var() # Média dos desvios ao quadrado
```

6. Medidas Separatizes (Posicionamento)

Para que serve: Dividir o conjunto de dados em partes proporcionais (Quartis, Decis e Percentis).

Por que é importante: Permite localizar a posição de um valor no ranking total (ex: o corte dos 1% maiores) e calcular o intervalo interquartil (IQR), fundamental para detectar Outliers.

```
Python
```

```
# Valor que separa os 50% inferiores dos 1% superiores
```

```
limite_inferior = df['Varavel'].quantile(0.99)
```

```
# Posição percentual de um valor alto (Scipy)
```

```
from scipy import stats
```

```
stats.percentileofscore(df['Varavel'], valor, alvo)
```

7. Visual Analytics: Estático vs. Interativo

Para que serve: Representar dados no espaço (estatística de quatro), Distplots (densidade) e Lineplots (series temporais).

Por que é importante: Boxplot são ferramentas prioritárias para comparar grupos (ex: Renda por UF), enquanto gráficos interativos facilitam a exploração de valores pontuais em grandes escala de tempo.

```
Python
```

```
# Seaborn: Boxplot Comparativo (Estático)
```

```
sns.boxplot(x='y', y='Categorial', data=df.query('Valor < 10000'), orient='h')
```

```
# Plotly: Linha Interativa
```

```
px.line(df_novo, x='Ano', y='Imigrante', markers=True)
```

8. Normalização e Reshaping (Melt)

Para que serve: Transformar tabelas largas (wide) em tabelas longas (long/tidy).

Por que é importante: A maioria das bibliotecas de programação moderna exige dados no formato "longo" para aplicar corrs e categorias (mas no Seaborn o corte no Pandas) de forma automática.

```
Python
```

```
d_longo = pd_pandas.reset_index().melt(id_vars='Ano', var_name='Pal', value_name='Valor')
```

1. Gráfico de Linhas (Line Plot)

Descrição: Conexão pontual entre dados individuais, mostrando a evolução de uma variável em relação a outra (geralmente tempo).

- Para que serve: Visualizar tendências, ciclos e flutuações em séries temporais.
- Por que é importante: Permite a fermentação primária para identificar se um fenômeno é estável, crescente, estagnado ou diminuindo ao longo dos anos.

```
Python
```

```
# Exemplo com Pandas para visualizar
```

```
df.plot(x='Ano', y='Populacao', color='blue', marker=True)
```

2. Gráfico de Barras (Bar Chart)

Descrição: Representa categorias no eixo X e volumes no eixo Y.

- Para que serve: Comparar grandezas entre diferentes grupos ou exibir distribuições.
- Por que é importante: Facilita a visualização de rankings (quem é o maior/menor) e a comparação direta entre categorias.

```
Python
```

```
# Exemplo de barras para distribuição de freqüências
```

```
df['Freq_Frequencia'].plot.bar(width=1, color='blue', alpha=0.2)
```

3. Histograma e Densidade (Histograma)

Descrição: Mostra a ocorrência de valores dentro de intervalos.

- Para que serve: Identificar o formato da distribuição (ex: Normal, Assimétrica) e o ponto de corte entre os quartis (inferior e superior).
- Por que é importante: Permite validar premissas estatísticas (como a normalidade) antes de aplicar algoritmos de Machine Learning.

```
Python
```

```
# Exemplo de barras para distribuição de freqüências
```

```
df['Corte'].hist(bins=10, color='blue')
```

4. Boxplot (Diagrama de Caixa)

Descrição: Representa a distribuição de dados através de quartis, ordenando a mediana (linha horizontal) e mostrando o intervalo interquartil (caixa) e os outliers (Pontos fora das hastas).

- Para que serve: Analisar a dispersão, simetria e identificar valores desviadores de norma.
- Por que é importante: É a melhor ferramenta para comparar distribuições entre grupos (ex: Renda por Sexo ou Cor), pois mostra não apenas a média, mas a variabilidade de cada grupo.

```
Python
```

```
# Boxplot comparativo com Seaborn
```

```
ax = sns.boxplot(x='Renda', y='UF', data=dados.query('Renda < 10000'), orient='h')
```

5. Medidas de Dispersão e Posição

Descrição: Além da tendência central (média/mediana), as medidas de dispersão indicam o grau de variabilidade entre os dados.

1. Varância (var)

- Para que serve: Mede o quanto cada valor dentro do conjunto está da média.
- Por que é importante: É a base para cálculos estatísticos avançados, embora sua unidade seja o quadrado da unidade original, o que dificulta a interpretação direta.

```
Python
```

```
variancia = dados.Renda.var() #
```

2. Desvio Padrão (std)

Descrição: Raiz quadrada da variancia, retornando a medida para a mesma unidade dos dados originais.

- Por que é importante: Indica a "variabilidade" ou "erro" espreado. Um desvio padrão baixo indica que os dados estão próximos da média; um alto indica grande dispersão.

```
Python
```

```
desvio_padrao = dados.Renda.std() #
```

3. Quartis e Percentis

Descrição: Dividir o conjunto de dados em partes iguais de 1% (percentis), 10% (decis) ou 25% (quartis).

- Por que é importante: Indica a "variabilidade" ou "erro" espreado. Um desvio padrão baixo indica que os dados estão próximos da média; um alto indica grande dispersão.

```
Python
```

```
# Valor que separa os 50% inferiores dos 1% superiores
```

```
limite_inferior = df['Varavel'].quantile(0.99)
```

```
# Posição percentual de um valor alto (Scipy)
```

```
from scipy import stats
```

```
stats.percentileofscore(df['Varavel'], valor, alvo)
```

1. Preparação e Estruturação do Ambiente

Descrição: Organizar as estruturas necessárias para manipulação de matrizes, estruturação de dados e execução de práticas.

- Imports Essenciais: Importação de pandas para cálculos, numpy para cálculos, e pandas para manipulação de dados.
- Carga e Indexação: Utilizado de arquivos CSV e a definição de uma coluna categoria para indexar facilmente buscas diretas e manipulações rápidas.
- Filtragem: Criação de listas de string para selecionar colunas específicas, como intervalos de anos.

2. Manipulação e Transformação de Dados

- Subsetting: Extrai de fatias específicas do dataset para isolar entidades (ex: um país ou uma classe social).
- Discretização: Utilizado para transformar dados numéricos brutos (como renda) em categorias ou faixas para facilitar a interpretação.
- Resampling (Melt): Conversão do formato de tabela "largo" para "largo", essencial para combinar dados com estruturas gráficas modernas que utilizam categorias para cores e

legendas.

3. Estatística Descritiva

A estatística descritiva serve para resumir e descrever as características de um conjunto de dados.

Medidas de Tendência Central

Descrição: Indicam a centralização da distribuição dos dados sobre diferentes perspectivas.

- Média: Centro de massa dos dados, sendo sensível a valores extremos (outliers).
- Mediana: Valor central que divide o conjunto de dados exatamente ao meio, sendo robusta contra outliers.
- Moda: O valor que ocorre com maior frequência na amostra.

Mean, Median and Mode

Descrição: Representa a centralização da amostra.

</

