Programação para Ciência de Dados

Apresentação da Disciplina e Fundamentos

Arthur Casals

07 de Outubro de 2025

Agenda

- Apresentação da disciplina
- Google Colab e Sintaxe Básica
- Estruturas de Controle
- Funções e Escopo
- Estrutura de Dados

Bloco 1

Apresentação da disciplina

Datas e Horários

Aulas expositivas

- Terças e quintas (08:00 às 11:00)
- Prof. Arthur Casals

Aulas práticas

- Terças e quintas (11:00 às 12:00) e Sextas (10:00 às 12:00)
- Auxiliadas por Victor Hugo Rocha (monitor)

Dinâmicas das aulas

Aulas expositivas

- Ciclos de exposição intercalados com pontos de checagem, demonstrações, e exercícios práticos
- Dúvidas via chat (gerenciadas pelo monitor)

Dinâmicas das aulas

Aulas práticas

- Após cada aula expositiva, 1h para exercícios práticos
- Sextas-feiras: exercícios mais complexos

Interação

- Todas as interações e entregas serão feitas via Zoom (aulas) e Moodle da disciplina
  - Moodle: `https://ead.fdte.org.br`

Avaliação

Critérios

- Participação (10% da nota)
- Proatividade (30% da nota)
- Conteúdo (60% da nota)

Avaliação

Participação (10% da nota)

- *Objetivo:* fazer com que o aluno participe das aulas práticas, onde os conceitos expostos serão aplicados.
- *Entrega:* ao final de cada exercício prático (terças e quintas) o aluno deverá entregar seu notebook. A entrega é feita via Moodle.
- *Avaliação:* nota igual à proporção de entregas feitas em relação ao total de entregas previstas.

Avaliação

Proatividade (30% da nota)

- Após cada aula expositiva, será disponibilizada (via Moodle) uma questão a ser entregue no início da aula expositiva seguinte, versando sobre o conteúdo dessa aula futura.
- *Objetivo:* fazer com que o aluno reflita sobre o conteúdo da aula seguinte.
- *Entrega:* a entrega será feita via Moodle, ao longo dos primeiros 30 minutos da aula expositiva.
- *Avaliação:* nota igual à proporção de entregas corretas feitas em relação ao total de entregas previstas. A correção será feita pelo monitor, que analisará a coerência da resposta dada em relação à questão proposta.

Avaliação

Conteúdo (60% da nota)

- A cada duas semanas, durante os exercícios práticos semanais (sextas- feiras), o notebook entregue será avaliado.
- *Objetivo:* avaliar a qualidade do conteúdo prático produzido pelo aluno ao longo do curso.
- *Entrega:* a entrega será feita via Moodle, nos dias 17/10, 31/10 e 14/11.
- *Avaliação:* A correção será feita pelo monitor, que avaliará os seguintes quesitos:
  - Corretude: correspondendo a 90% da nota, determina se o aluno respondeu ao requisitado durante a tarefa.
  - Organização e documentação do código:correspondendo a 10% da nota, avalia comentários e a organização geral do código produzido.

Avaliação

Entregas de trabalhos

- Entregues sempre ao final das aulas práticas, via Moodle.
- **Todas as entregas são individuais.**

Presença

- No início de cada aula expositiva (terças e quintas)
- No início da aula prática semanal (sexta)
- **Obrigatoriamente** nos primeiros 30 minutos de aula

Aprovação

- Presença: frequência mínima de 75%
- Notas: média final mínima 5,0

Dúvidas

- Terças e quintas: no início das aulas **práticas**, com duração máxima de 30 minutos.
- Sextas: no início das aulas, com duração máxima de 45 minutos.

Programa de aulas

- **Semana 1:** Python e Estruturas de Dados
- **Semana 2:** NumPy
- **Semana 3:** Pandas
- **Semana 4:** Análise Exploratória de Dados (EDA)
- **Semana 5:** Pré-processamento de Dados
- **Semana 6:** Séries Temporais e Dados Multidimensionais

Programa de aulas - Semana 1

- **Aula 1:** Apresentação da Disciplina e Fundamentos
  - Apresentação da disciplina
  - Google Colab e Sintaxe Básica
  - Estruturas de Controle
  - Funções e Escopo
  - Estrutura de Dados
- **Aula 2:** Estruturas de Dados Avançadas e Algoritmos Básicos
  - Estruturas aninhadas e coleções
  - Compreensões avançadas e geradores
  - Algoritmos de busca e ordenação
  - Complexidade e desempenho

Programa de aulas - Semana 2

- **Aula 3:** Introdução ao NumPy
  - Fundamentos
  - Indexação e fatiamento
  - Operações vetorizadas
  - Estatísticas e agregações
- **Aula 4:** NumPy Avançado: Matrizes e Álgebra Linear
  - Arrays multidimensionais
  - Operações matriciais
  - Álgebra linear com np.linalg
  - Concatenação, stacking e I/O

Programa de aulas - Semana 3

- **Aula 5:** Introdução ao Pandas
  - Fundamentos
  - Visualização e inspeção de dados
  - Seleção e indexação
  - Filtragem e ordenação
- **Aula 6:** Manipulação de Dados com Pandas
  - Transformações
  - Agregações
  - Combinações de dataframes
  - Reshape e Pivot

Programa de aulas - Semana 4

- **Aula 7:** Análise Exploratória de Dados (EDA) - Parte 1
  - Estatística descritiva
  - Correlação e relações
  - Visualização com Matplotlib
  - Múltiplos gráficos
- **Aula 8:** Análise Exploratória de Dados (EDA) - Parte 2
  - Visualização com Seaborn
  - Análise de variáveis categóricas
  - Análise multivariada
  - Storytelling com dados

Programa de aulas - Semana 5

- **Aula 9:** Pré-processamento de Dados - Limpeza e Tratamento
  - Dados Faltantes
  - Técnicas de Imputação
  - Limpeza de Dados
  - Validação e Consistência
- **Aula 10:** Pré-processamento de Dados - Transformação e Normalização
  - Feature Engineering
  - Encoding de Variáveis Categóricas
  - Normalização e Padronização
  - Transformações Matemáticas

Programa de aulas - Semana 6

- **Aula 11:** Séries Temporais - Fundamentos e Análise
  - Fundamentos de Séries Temporais
  - Resample e Agregação Temporal
  - Rolling Windows e Médias Móveis
  - Decomposição e Análise
- **Aula 12:** Dados Multidimensionais
  - Dados Multidimensionais
  - Visualização de Alta Dimensão
  - Redução de Dimensionalidade

Materiais do Curso

Por aula, quando aplicável:

- Notebook Principal (teoria + exemplos + exercícios guiados)
- Notebook de Prática Independente
- Notebook de Soluções (gabarito)
- Datasets específicos
- Material complementar (referências, exercícios de fixação, notebooks)

Observações Finais

- Todas as aulas utilizam Google Colab como ambiente
- Ênfase em datasets reais e problemas práticos
- Progressão gradual de complexidade

Recursos e Material Utilizado

Semana 1:

- Documentação Python: `https://docs.python.org/pt-br/3/`
- Tutorial estruturas de dados: `https://docs.python.org/pt-br/3/tutorial/datastructures.html`

Recursos e Material Utilizado

- **Datasets:**
  - Titanic (Aulas 1-6): `https://www.kaggle.com/c/titanic`
- **Ferramentas:** Google Colab, Python 3.11+, Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn.

Nota Importante

Os links para os datasets utilizados nas aulas 7-12 serão disponibilizados posteriormente.

Bloco 2

Google Colab e Sintaxe Básica

Introdução ao Google Colab

O que é:

- Ambiente de notebook Jupyter gratuito na nuvem
- GPU e TPU gratuitas para computação pesada
- Integração nativa com Google Drive e GitHub
- Python pré-instalado com principais bibliotecas
- Colaboração em tempo real como Google Docs

Por que usar no curso:

- Zero configuração - funciona no browser
- Acesso universal - qualquer dispositivo
- Backup automático no Google Drive
- Compartilhamento fácil de notebooks
- Hardware potente sem custo

Introdução ao Google Colab

- Acesso direto: `https://colab.research.google.com`
- Via Google Drive
  - Novo -> Mais -> Google Colaboratory
  - Se não aparecer: ”Conectar mais aplicativos”
- Requisitos:
  - Conta Google (Gmail)
  - Navegador atualizado (Chrome recomendado)
  - Conexão com internet estável

Live Coding

Demonstração Prática
Duração: 30 minutos

Entrega

⚠ Atenção

Use o Moodle para entregar o notebook gerado.
Limite de entrega: até o final da aula.

Bloco 3

Estruturas de Controle

Variáveis: Guardando Informações na Memória

O que são variáveis?

- ”Caixas”que guardam valores na memória
- Têm um nome para identificá-las
- Podem mudar de valor durante a execução

Características do Python:

- **Tipagem dinâmica:** Não precisa declarar o tipo
- **Case sensitive:** `nome` $\neq$ `Nome` $\neq$ `NOME`
- Podem mudar de valor durante a execução

Boas práticas para nomes:

- Use letras, números e underscore (_)
- Comece com letra ou underscore
- Evite palavras reservadas (if, for, while...)
- Use snake_case: `idade_usuario`, `taxa_conversao`

Exemplo de uso de variáveis (I)

Python

```python
# Diferentes tipos de dados
nome_aluno = "Ana Silva"           # String (texto)
idade = 23                         # Integer (inteiro)
altura = 1.68                      # Float (decimal)
aprovado = True                    # Boolean (verdadeiro/falso)
salario = None                     # None (valor nulo)

# Exibindo os valores
print("=== DADOS DO ALUNO ===")
print(f"Nome: {nome_aluno}")
print(f"Idade: {idade} anos")
print(f"Altura: {altura}m")
print(f"Aprovado: {aprovado}")
print(f"Salário: {salario}")

```

Exemplo de uso de variáveis (II)

Python

```python
# Verificando os tipos
print("\n=== TIPOS DE DADOS ===")
print(f"tipo de 'nome_aluno': {type(nome_aluno)}")
print(f"tipo de 'idade': {type(idade)}")
print(f"tipo de 'altura': {type(altura)}")
print(f"tipo de 'aprovado': {type(aprovado)}")

```

Operadores Aritméticos: A Base dos Cálculos

Operadores Básicos:

- `+` **Adição:** `5 + 3 = 8`
- `-` **Subtração:** `10 - 4 = 6`
- `*` **Multiplicação:** `7 * 2 = 14`
- `/` **Divisão:** `15 / 4 = 3.75` (sempre retorna float)

Operadores Especiais:

- `//` **Divisão inteira:** `15 // 4 = 3` (sem casas decimais)
- `%` **Módulo (resto):** `15 % 4 = 3` (resto da divisão)
- `**` **Potência:** `2 ** 3 = 8` (2 elevado a 3)

Nota Importante

Precedência: Python segue a ordem matemática tradicional (parênteses, potência, multiplicação/divisão, soma/subtração)

Operadores de Comparação: Tomando Decisões

Operadores de Comparação:

- == **Igual a:** 5 == 5 → `True`
- *!*= **Diferente de:** 5 != 3 → `True`
- > **Maior que:** `10 > 5` → `True`
- < **Menor que:** `3 < 8` → `True`
- >= **Maior ou igual:** 5 >= 5 → `True`
- <= **Menor ou igual:** 4 <= 7 → `True`

Cuidado Comum:

- Use == para comparação, não =
- = é para atribuição: `x = 5`
- == é para comparação: x == 5

Resultado sempre é `True` ou `False`

Operadores Lógicos: Combinando Condições

`and` (E) - Ambas condições devem ser verdadeiras

- `True and True` → `True`
- `True and False` → `False`
- `False and False` → `False`

`or` (OU) - Pelo menos uma condição deve ser verdadeira

- `True or True` → `True`
- `True or False` → `True`
- `False or False` → `False`

`not` (NÃO) - Inverte o valor lógico

- `not True` → `False`
- `not False` → `True`

Estruturas de Controle: Tomando Decisões

O que são Estruturas de Controle?

- Comandos que alteram o fluxo de execução do programa
- Permitem tomar decisões baseadas em condições
- Fazem o código repetir ações ou pular etapas

Tipos Principais:

1. **Condicionais** (`if`, `elif`, `else`) - Decisões
2. **Loops** (`for`, `while`) - Repetições
3. **Controle de fluxo** (`break`, `continue`) - Interrupções

Analogia da Vida Real:
SE está chovendo: leve guarda-chuva
SENÃO SE está frio: leve casaco
SENÃO: vá de camiseta

Estrutura condicional

Estrutura Básica:

```
if condicao:
    # codigo executado se condicao for True
    comando1
    comando2

```

Estrutura Completa:

```
if condicao1:
    # executa se condicao1 for True
elif condicao2:
    # executa se condicao1 for False e condicao2 for True
elif condicao3:
    # executa se condicoes anteriores forem False
else:
    # executa se TODAS as condicoes anteriores forem False

```

Estrutura condicional

Pontos Importantes:

- Indentação é obrigatória (4 espaços ou 1 tab)
- Dois pontos (:) após cada condição
- `elif` = ”else if”(senão se)

Loops: Automatizando Repetições

Por que Loops são Fundamentais?

Problema Comum:

- Sem loops: código repetitivo e ineficiente
- Com loop: elegante e eficiente

Vantagens dos Loops:

- **DRY Principle:** Don't Repeat Yourself
- **Flexibilidade:** Fácil mudar quantas repetições
- **Escalabilidade:** Funciona para 5 ou 5000 repetições
- **Manutenibilidade:** Mudança em um lugar só

Tipos de Loops em Python:

1. `for` - Número conhecido de repetições
2. `while` - Repetir enquanto condição for verdadeira

Loop for: Iterando com Estilo

Sintaxe e Conceitos do for

Estrutura Básica:

```
for variavel in sequencia:
    # codigo a ser repetido
    comando1
    comando2

```

Tipos de Sequências:

- **range():** `for i in range(5)` → 0, 1, 2, 3, 4
- **Listas:** `for item in [1, 2, 3]` → 1, 2, 3
- **Strings:** `for char in "Python"` → P, y, t, h, o, n

Função range():

- `range(5)` → 0, 1, 2, 3, 4 (0 até 4)
- `range(1, 6)` → 1, 2, 3, 4, 5 (1 até 5)
- `range(0, 10, 2)` → 0, 2, 4, 6, 8 (de 2 em 2)

Repetições

Python

```python
# Exemplo 1: Contagem simples
print("=== CONTAGEM DE 1 A 5 ===")
for numero in range(1, 6):
    print(f"Contando: {numero}")

# Exemplo 2: Iterando sobre lista
print("\n=== FRUTAS DISPONIVEIS ===")
frutas = ["maca", "banana", "laranja"]
for fruta in frutas:
    print(f"Fruta: {fruta.capitalize()}")

```

Repetir Enquanto Condição

Python

```python
# Loop while: executa enquanto condicao for True
print("=== CONTAGEM COM WHILE ===")
contador = 1
while contador <= 5:
    print(f"Contador: {contador}")
    contador += 1  # Incremento

```

Controle de Fluxo

`break` - Interrompe o loop completamente

```
for i in range(10):
    if i == 5:
        break  # Para o loop aqui
    print(i)  # Imprime 0, 1, 2, 3, 4

```

`continue` - Pula para próxima iteração

```
for i in range(5):
    if i == 2:
        continue  # Pula quando i=2
    print(i)  # Imprime 0, 1, 3, 4

```

Casos de Uso:

- **break:** Encontrar item específico, sair de loops infinitos
- **continue:** Pular valores inválidos, filtrar dados

Bloco 4

Funções e Escopo

Por que Usar Funções?

Problemas sem funções:

- Código repetitivo e difícil de manter
- Dificulta reutilização de lógica
- Mais propenso a erros

Benefícios das funções:

- **Reutilização:** Escreva uma vez, use várias vezes
- **Organização:** Código mais limpo e estruturado
- **Manutenção:** Correções em um só lugar
- **Abstração:** Esconde complexidade
- **Testabilidade:** Mais fácil testar partes isoladas

Funções: Sintaxe Básica

</> Python

```python
def calcular_area_retangulo(largura, altura):
    """Calcula a area de um retangulo."""
    area = largura * altura
    return area

# Usando a funcao
area1 = calcular_area_retangulo(5, 3)
area2 = calcular_area_retangulo(8, 4)
area3 = calcular_area_retangulo(10, 2)

print(f"Area 1: {area1}")  # Area 1: 15
print(f"Area 2: {area2}")  # Area 2: 32
print(f"Area 3: {area3}")  # Area 3: 20

```

Funções: Sintaxe Básica

Componentes:

- `def`: palavra-chave para definir função
- `largura, altura`: parâmetros
- `return`: valor retornado

Componentes de uma Função

```python
def nome_da_funcao(parametro1, parametro2):
    """
    Docstring: descricao da funcao
    Args: parametros esperados
    Returns: o que a funcao retorna
    """
    # Corpo da funcao
    resultado = parametro1 + parametro2
    return resultado

# Chamada da funcao
valor = nome_da_funcao(10, 20)  # argumentos
print(valor)  # 30

```

Componentes de uma Função

Terminologia importante:

- **Parâmetros:** variáveis na definição da função
- **Argumentos:** valores passados na chamada
- **Return:** palavra-chave para retornar valores

Bloco 5

Estrutura de Dados

Listas: Coleções Ordenadas de Dados

- Uma lista em Python é uma estrutura de dados que permite armazenar uma coleção de itens em uma única variável, podendo ser de diferentes tipos (números, strings, até outras listas).
- As listas são mutáveis: é possível alterar valores, adicionar e remover elementos após a criação.
- Suportam ordem: cada elemento da lista tem uma posição definida (índice), começando em 0; a ordem é sempre preservada.
- Permitem elementos duplicados, ou seja, podem conter itens com o mesmo valor.

Listas: Coleções Ordenadas de Dados

- Sintaxe: são definidas entre colchetes, por exemplo: `[10, 'Python', 3.5]`.
- Suportam várias operações como percorrer com loops (for), fatiamento (`lista[1:3]`), métodos como `append()`, `remove()`, `sort()`, entre outros.
- Aceitam aninhamento: listas podem conter outras listas como elementos, formando estruturas mais complexas (ex: matrizes/bidimensionais).
- São amplamente usadas para organizar, processar, filtrar, ordenar e manipular coleções de dados em Python.

Listas: Coleções Ordenadas de Dados

Python

```python
# Criando listas
numeros = [1, 2, 3, 4, 5]
nomes = ["Ana", "Bruno", "Carlos"]
misto = [1, "Python", 3.14, True]
lista_vazia = []

# Acessando elementos (indice comeca em 0)
print(numeros[0])    # 1 (primeiro elemento)
print(nomes[1])      # Bruno (segundo elemento)
print(numeros[-1])   # 5 (ultimo elemento)
print(nomes[-2])     # Bruno (penultimo elemento)

# Informacoes sobre a lista
print(len(numeros))  # 5 (tamanho da lista)

```

Manipulando Listas

Python

```python
frutas = ["maca", "banana", "laranja"]

# Adicionando elementos
frutas.append("uva")          # Adiciona ao final
print(frutas)  # ['maca', 'banana', 'laranja', 'uva']

frutas.insert(1, "manga")     # Adiciona na posicao 1
print(frutas)  # ['maca', 'manga', 'banana', 'laranja', 'uva']

```

Manipulando Listas

Python

```python
# Removendo elementos
# ['maca', 'manga', 'banana', 'laranja', 'uva']
frutas.remove("banana")       # Remove por valor
print(frutas)  # ['maca', 'manga', 'laranja', 'uva']

fruta_removida = frutas.pop() # Remove ultimo e retorna
print(fruta_removida)         # uva

# Modificando elementos
frutas[0] = "pera"
print(frutas)  # ['pera', 'manga', 'laranja']

```

Tuplas: Listas Imutáveis

- Tupla é uma estrutura de dados em Python usada para agrupar múltiplos elementos em uma única variável, assim como as listas.
- A principal característica de tuplas é a imutabilidade: uma vez criada, seus elementos não podem ser modificados, adicionados ou removidos. O tamanho da tupla também não muda.
- São definidas usando parênteses: por exemplo, `(10, 'Python', 3.5)`.

Tuplas: Listas Imutáveis

- Tuplas podem armazenar tipos diferentes de dados e podem ser aninhadas (tuplas dentro de tuplas).
- Permitem acesso por índice (posição), e suportam operações como fatiamento e iteração em loops, exatamente como listas.
- Ideal para representar informações que não devem ser alteradas, oferecendo segurança e melhor desempenho em algumas operações por conta da imutabilidade.
- São utilizadas com frequência para retornos múltiplos de funções, agrupamento de dados fixos, e outros contextos onde a integridade é desejada.

Tuplas: Listas Imutáveis

</> Python

```python
# Criando tuplas
coordenadas = (10, 20)
cores_rgb = (255, 128, 0)
dados_pessoa = ("Ana", 25, "Engenheira")
# Tupla de um elemento (virgula é obrigatoria!)
tupla_um = (42,)  # Correto
tupla_um = (42)   # Isso é apenas um numero!
# Acessando elementos (igual as listas)
print(coordenadas[0])     # 10
print(dados_pessoa[-1])   # Engenheira
# Tentativa de modificacao gera erro
# coordenadas[0] = 15  # TypeError

```

Hashtables

- Uma hashtable (ou tabela hash) é uma estrutura de dados que armazena pares de chave-valor.
- Para guardar e localizar rapidamente valores, a hashtable usa uma função hash: ela transforma a chave (por exemplo, um nome, número ou string) em um número inteiro, chamado de índice.
- Esse índice indica a posição no array (tabela) onde o valor será armazenado ou buscado.
- É muito eficiente: operações como inserir, buscar e remover são rápidas.
- Exemplos de uso: guardar idades de pessoas por nome, contar frequência de palavras, implementar dicionários em linguagens de programação.

Dicionários: Dados Organizados

- Um dicionário é uma coleção não ordenada de pares `chave: valor`, onde cada chave é única.
- Pode ser criado usando ou a função `dict()`, por exemplo: `d = 'nome': 'Alice', 'idade': 25`.
- Chaves podem ser de tipos imutáveis (strings, números, tuplas), e valores podem ser de qualquer tipo.
- Permite acesso rápido aos valores a partir das chaves, sem depender de ordem ou índices como nas listas.
- Dicionários são **mutáveis**, ou seja, é possível adicionar, modificar ou remover pares.

Dicionários: Dados Organizados

Python

```python
# Criando dicionarios
pessoa = {
    "nome": "Maria",
    "idade": 28,
    "profissao": "Data Scientist",
    "ativo": True
}

# Acessando valores
print(pessoa["nome"])       # Maria
print(pessoa.get("idade"))  # 28

```

Dicionários: Dados Organizados

</> Python

```python
"""(continuação)"""
# Modificando e adicionando
pessoa["idade"] = 29              # Modifica valor
pessoa["cidade"] = "Sao Paulo"    # Adiciona nova chave

print(pessoa)

```

Chaves devem ser únicas e imutáveis (strings, números, tuplas)

Estruturas de Dados - Quando Usar Cada Uma?

Estrutura	Quando Usar	Exemplo
Lista	Sequência ordenada, elementos podem repetir	`notas = [8.5, 7.0, 9.2]`
Tupla	Dados fixos, não mudam	`ponto = (10, 20)`
Dicionário	Dados relacionados, acesso por chave	`pessoa = {"nome": "Ana"}`

Revisão dos Conceitos da Aula

Fundamentos Python:

- Variáveis e tipos de dados (int, float, string, bool)
- Operadores (aritméticos, comparação, lógicos)
- Estruturas condicionais (if, elif, else)

Estruturas de Repetição:

- Loop for com range() e listas
- Loop while com condições
- Controle de fluxo (break, continue)

Revisão dos Conceitos da Aula

Funções:

- Definição com def
- Parâmetros e argumentos
- Return vs print

Estruturas de Dados:

- Listas (mutáveis, ordenadas)
- Tuplas (imutáveis, ordenadas)
- Dicionários (chave-valor)

Introdução ao Dataset Titanic

O que vamos descobrir:

- Quantos passageiros sobreviveram?
- Qual classe teve maior taxa de sobrevivência?
- Idade influenciou na sobrevivência?
- Homens ou mulheres tiveram mais chance?

Estrutura dos dados:

- **PassengerId:** ID do passageiro
- **Survived:** 0 = não sobreviveu, 1 = sobreviveu
- **Pclass:** Classe do bilhete (1, 2, 3)
- **Name:** Nome do passageiro
- **Sex:** Sexo (male/female)
- **Age:** Idade em anos
- **Fare:** Tarifa paga

A Seguir: Aula Prática

✋ Exercício Prático

Tempo: 60 minutos
Entrega: via Moodle (notebook)
Tarefa:

1. (o enunciado será atualizado durante a aula)

Obrigado