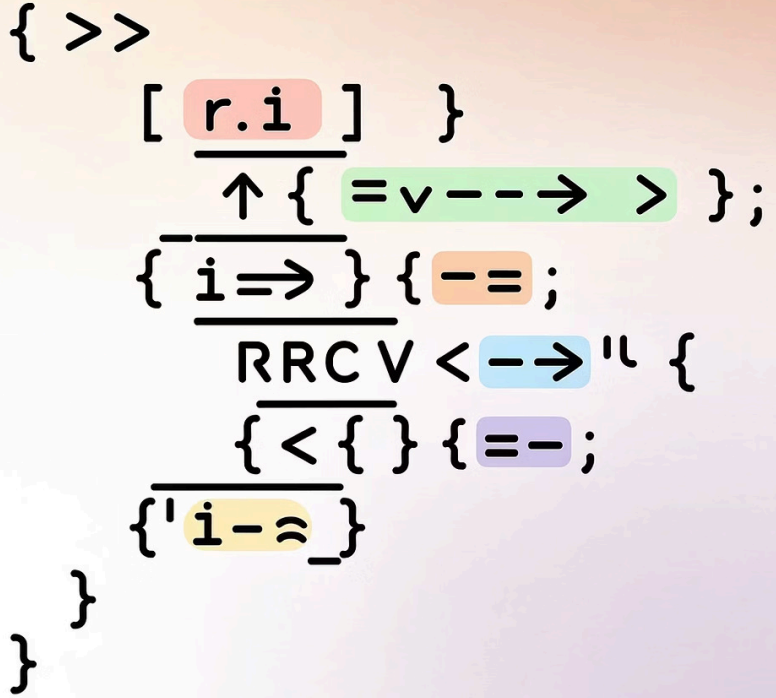


CEFET/RJ

Objetos em Python

Python trabalha com **objetos** que representam dados na memória. Uma variável é um nome que aponta para um objeto. Objetos podem guardar números, textos ou estruturas mais complexas.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br

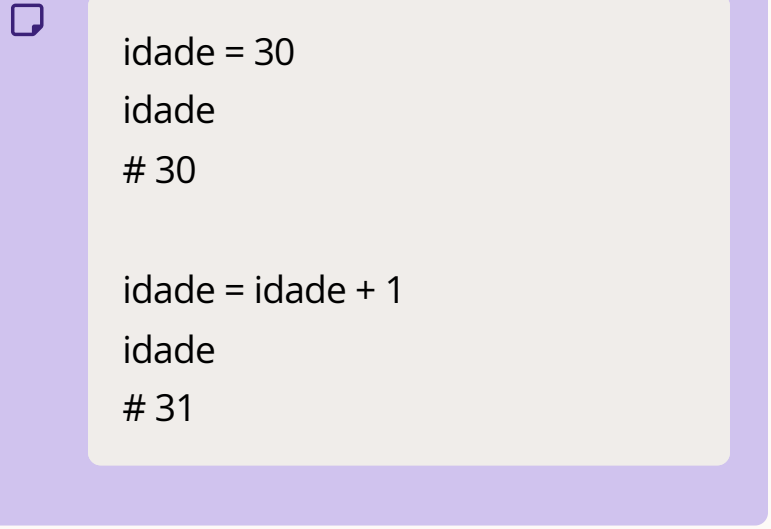
https://eic.cefet-rj.br/~eogasawara

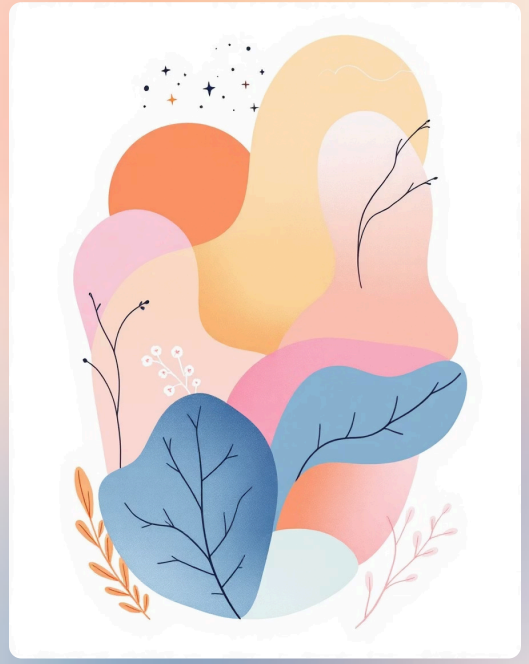
CONCEITOS FUNDAMENTAIS

O que é uma Variável

Uma variável é um **nome associado a um valor**. Esse valor fica armazenado na memória do computador. O nome permite acessar, reutilizar e modificar o dado.

Quando atribuímos um valor a uma variável, estamos criando uma referência na memória que podemos usar sempre que precisarmos daquele dado.





Objetos, Tipos e Valores

Valor

O conteúdo armazenado pelo objeto

Tipo

Define quais operações podem ser feitas

Operações

Funções que produzem novos objetos

Todo objeto em Python possui um **valor** e um **tipo**. O tipo determina o comportamento do objeto e quais operações são permitidas sobre ele.

```
a = 5
b = 2
c = a * b
type(c)
# <class 'int'>
```

Data Frames (Tabelas)

Data Frames são estruturas tabulares fundamentais para análise de dados. Eles combinam várias colunas em uma única estrutura, onde cada coluna pode ter um tipo de dado diferente. São amplamente utilizados em ciência de dados para organizar, manipular e analisar informações de forma eficiente.

Estrutura organizada

Linhas representam registros individuais, colunas representam atributos ou características

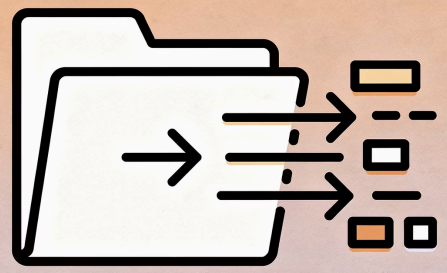
Tipos flexíveis

Cada coluna pode conter strings, números, datas ou outros tipos de dados

Análise poderosa

Permite filtrar, agrupar, calcular estatísticas e visualizar dados rapidamente

```
df = pd.DataFrame({
 "face": ["ás", "dois", "quatro"],
 "naipe": ["ouros", "copas", "paus"],
 "valor": [1, 2, 4]
})
print(df)
```



Salvando e Lendo Arquivos

Uma das grandes vantagens dos Data Frames é a capacidade de persistir dados em arquivos. Tabelas podem ser facilmente exportadas para o formato CSV (Comma-Separated Values), que é amplamente compatível e legível. Posteriormente, esses arquivos podem ser lidos novamente, permitindo compartilhar dados com colegas, fazer backup ou continuar análises em outro momento.

Salvar em CSV

```
df.to_csv("cartas.csv",
        index=False)
```

Ler do CSV

```
cartas = pd.read_csv(
    "cartas.csv")
```

Compartilhar

Dados persistidos podem ser facilmente compartilhados entre projetos e colaboradores

Valores Ausentes e Especiais

Nem sempre temos todos os dados completos. Python oferece diferentes maneiras de representar a ausência de informação, cada uma com seu propósito específico e comportamento particular em cálculos e operações.

None

Indica a ausência completa de um objeto em Python. É o valor nulo padrão da linguagem e pode ser usado em qualquer contexto.

```
x = [1, None, 3]
```

np.nan

Representa especificamente um número ausente (Not a Number). É usado em arrays NumPy e cálculos numéricos quando um valor está faltando.

```
y = np.array([1.0, np.nan, 3.0])
```

Importante: `None` e `np.nan` não são intercambiáveis. Use `None` para objetos Python genéricos e `np.nan` para operações numéricas com NumPy.

Lidando com Valores Ausentes

Valores ausentes podem "contaminar" cálculos estatísticos, retornando resultados inesperados. Felizmente, Python oferece funções especializadas que sabem como ignorar valores inválidos e trabalhar apenas com os dados válidos disponíveis.

⚠️ Problema

```
y = [1.0, np.nan, 3.0]
np.mean(y)
# Resultado: nan
```

A função padrão retorna `nan` quando encontra valores ausentes, invalidando todo o cálculo.

✓ Solução

```
np.nanmean(y)
# Resultado: 2.0
```

Funções com prefixo `nan` ignoram valores ausentes e calculam corretamente a média dos valores válidos.

Identificar valores ausentes

```
np.isnan(y)
# [False, True, False]
```

Usar funções nan-aware

Use `nanmean`, `nansum`, `nanstd` para cálculos robustos

Remover ou preencher valores

Opcionalmente, remova linhas com `dropna()` ou preencha com fillna()

O que são Objetos

Dados Simples

Números inteiros, decimais e textos que representam valores únicos

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

Estruturas

Listas, dicionários e outras coleções que agrupam múltiplos valores

```
lista = [1, 2, 3]
type(lista)
# <class 'list'>
```

Comportamento

Cada tipo tem operações específicas e características próprias

```
texto = "dados"
texto.upper()
# 'DADOS'
```

Em Python, variáveis armazenam objetos de vários tipos. Cada tipo tem um comportamento específico que define como podemos trabalhar com ele.

ARRAYS NUMPY

Vetores (Sequências Numéricas)

Sequências numéricas são a **base de muitas operações** em programação científica e análise de dados.

Em Python, arrays do NumPy são usados para armazenar vários valores do mesmo tipo de forma eficiente.

```
import numpy as np

dado = np.array([1, 2, 3, 4, 5, 6])
dado
# array([1, 2, 3, 4, 5, 6])
```

Vetores Unitários

Valor Único

Um único valor pode ser tratado como uma sequência de tamanho 1

Consistência

Isso torna o comportamento das operações mais previsível

Verificação

Podemos sempre verificar o tamanho com `len()`

```
import numpy as np

numero = np.array([5])
len(numero)
# 1
```

```
dado = np.array([1, 2, 3, 4, 5, 6])
len(dado)
# 6
```

Tipo Inteiro e Texto

Inteiros (int)

Números inteiros ocupam menos memória que números decimais e são usados para contagens e índices

```
inteiro = 1
type(inteiro)
# <class 'int'>
```

Textos (str)

Strings são definidas entre aspas e podem conter qualquer caractere

```
texto = "ás"
type(texto)
# <class 'str'>
```

Python distingue claramente entre números inteiros e textos. Cada tipo tem operações específicas e comportamentos diferentes.

TRABALHANDO COM SEQUÊNCIAS

Sequências e Funções

Sequências podem armazenar números ou textos. Funções especializadas operam sobre essas sequências, permitindo análises e transformações dos dados.

Sequências Numéricas

```
import numpy as np
cartas = np.arange(1, 14)
np.sum(cartas)
# 91
```

Funções matemáticas retornam valores agregados como soma, média e máximo

Sequências de Texto

```
faces = np.array(["ás", "dois", "três",
  "quatro", "cinco", "seis", "sete",
  "oito", "nove", "dez", "valete",
  "dama", "rei"])
faces.max()
# 'três'
```

Textos também podem ser ordenados e analisados

Tipo Decimal (float)

Em Python, números decimais usam **ponto flutuante**. Eles permitem maior precisão em cálculos científicos e financeiros.

Mesmo números que parecem inteiros podem ser armazenados como float, dependendo de como são criados ou do contexto de uso.

```
dado = np.array([1, 2, 3, 4, 5, 6],
 dtype=float)
dado
# array([1., 2., 3., 4., 5., 6.])

dado.dtype
# dtype('float64')
```

✓ VALORES LÓGICOS

Tipo Lógico (Booleano)

Comparações

Operações de comparação retornam True ou False

Decisões

Valores booleanos controlam fluxo de execução

Filtros

Permitem selecionar subconjuntos de dados

Valores lógicos representam **verdadeiro ou falso**. Eles são resultado de comparações e essenciais para decisões e filtros em programação.

```
logico = np.array([True, False, 3 >= 4, 3 < 4, 3 != 4, 4 == 4])
logico
# array([ True, False, False, True, True, True])
```

Números Complexos

Python suporta **números complexos** nativamente. Eles possuem parte real e parte imaginária, representados com o sufixo `j`.

São amplamente usados em ciência, engenharia, processamento de sinais e física quântica.

```
comp = np.array([1+1j, 1+2j, 1+3j])
comp
# array([1.+1.j, 1.+2.j, 1.+3.j])

comp.dtype
# dtype('complex128')
```

DADOS BINÁRIOS

Tipo Byte (Dados Brutos)

Representação

Bytes representam dados binários de baixo nível, essenciais para trabalhar com arquivos e comunicação

Intervalo

Cada byte pode assumir valores entre 0 e 255 (8 bits)

Tratamento

Valores fora do intervalo geram erro ou precisam ser convertidos explicitamente

```
r = bytearray(3)
r[1] = 255
# r[2] = 1024 # gera erro: valor fora do intervalo
r[2] = 1024 % 256 # conversão explícita para 0
r
# bytearray(b'\x00\xff\x00')
```

Atributos dos Objetos

Objetos podem ter **metadados associados** que fornecem informações adicionais sobre os dados. Em arrays, rótulos podem ser simulados com dicionários, facilitando a interpretação.

Criando Dados Rotulados

```
dado = np.array([1, 2, 3, 4, 5, 6])
nomes = ["um", "dois", "três",
         "quatro", "cinco", "seis"]
rotulado = dict(zip(nomes, dado))
```

Resultado

```
rotulado
# {'um': 1, 'dois': 2, 'três': 3,
#  'quatro': 4, 'cinco': 5,
#  'seis': 6}
```

Esses metadados ajudam a dar significado aos dados e tornam o código mais legível.

MATRIZES

Transformando em Matriz

Preenchimento por Coluna

No R, vetores viram matrizes preenchidas por coluna. No NumPy isso deve ser especificado com `order="F"`

Reshape Explícito

A função `reshape()` reorganiza os dados na forma desejada

```
import numpy as np
dado = np.arange(1, 7)
# preenchimento por coluna (equivalente ao R)
m = dado.reshape((2, 3), order="F")
m
# array([[1, 3, 5],
# [2, 4, 6]])
```

Criando Matrizes

A forma e a ordem de preenchimento **mudam o significado dos dados**. Podemos escolher entre preenchimento por linha ou por coluna.

Por Coluna (order="F")

```
dado = np.arange(1, 7)
por_coluna = dado.reshape((2, 3),
                          order="F")
por_coluna
# array([[1, 3, 5],
#        [2, 4, 6]])
```

Por Linha (order="C")

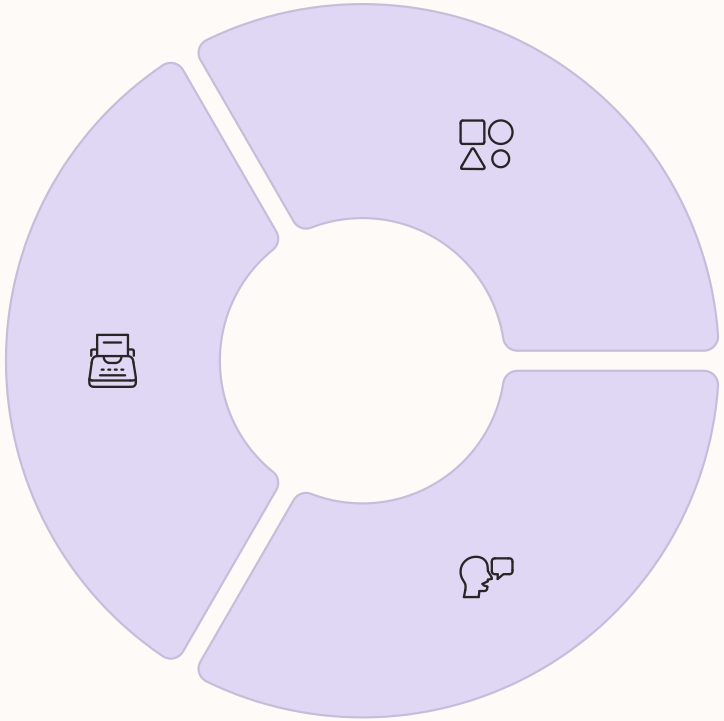
```
por_linha = dado.reshape((2, 3),
 order="C")
por_linha
# array([[1, 2, 3],
# [4, 5, 6]])
```

Tipo e Classe dos Objetos

Tipo Interno

Define a estrutura básica do objeto

```
type(dado)
# <class 'numpy.ndarray'>
```



Forma

Influencia como os dados são usados e acessados

```
dado.shape
# (2, 3)
```

Comportamento

A estrutura define as operações disponíveis

Um objeto possui um tipo interno e um comportamento específico. Em arrays, a forma determina como os dados são organizados e manipulados.

```
dado = np.arange(1, 7).reshape((2, 3))
```

TEMPORALIDADE

Data e Hora

Python possui tipos específicos para **datas e horas**. Eles representam instantes no tempo com precisão e são fundamentais em séries temporais, registros e análises cronológicas.

O módulo `datetime` fornece classes completas para manipular datas, horas, fusos horários e intervalos de tempo.

```
from datetime import datetime

now = datetime.now()
now
# datetime.datetime(2024, 1, 15,
# 14, 30, 45, 123456)

type(now)
# <class 'datetime.datetime'>
```

Dados Categóricos

Representação

Dados categóricos representam **classes ou grupos** fixos, como gênero, região ou categoria de produto

Análise

Facilitam análises estatísticas e visualizações por grupo

Eficiência

Ocupam menos memória que texto simples ao reutilizar categorias

```
import pandas as pd

genero = pd.Categorical(["feminino", "masculino", "feminino", "masculino"])
genero
# ['feminino', 'masculino', 'feminino', 'masculino']
# Categories (2, object): ['feminino', 'masculino']
```

FLEXIBILIDADE DE TIPOS

Conversões de Tipo

Objetos podem ser **convertidos entre tipos** diferentes. Isso permite combinar dados de origens variadas e realizar operações específicas. Algumas conversões acontecem automaticamente, enquanto outras precisam ser explícitas.

1

Booleano → Inteiro

```
int(True)   # 1
int(False)  # 0
```

0.0

Booleano → Decimal

```
float(False)  # 0.0
float(True)   # 1.0
```

1

Inteiro → Booleano

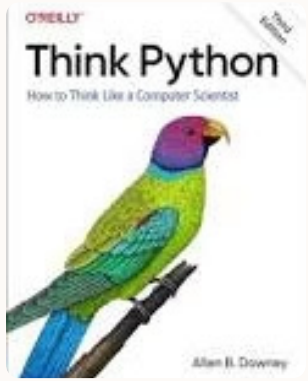
```
bool(1)  # True
bool(0)  # False
```

1

Qualquer → Texto

```
str(1) # '1'
str(True) # 'True'
```

Referências



Think Python

Downey, A. *Think Python: How to Think Like a Computer Scientist.* O'Reilly Media.

An essential introduction to programming fundamentals and computational thinking using Python.

Python Data Science Handbook

VanderPlas, J. *Python Data Science Handbook.* O'Reilly Media.

A comprehensive guide to essential tools for working with data in Python, including NumPy, Pandas, and visualization libraries.

Data Science from Scratch

Grus, J. *Data Science from Scratch.* O'Reilly Media.

Learn data science fundamentals by building algorithms and tools from the ground up using Python.