Técnicas de Programação

Fatec 2025

O que é um bom código?

- Aquele que executa mais rápido?
- Aquele que é mais fácil de ler?
- Aquele com de fácil manutenção?
- Cinco categorias:
  - Simplicidade
  - Modularidade
  - Legibilidade
  - Desempenho
  - robustez

Um bom código é importante

- quando o código é integrado a um sistema maior e/ou incorporado a um modelo de *machine learning* em produção
- A medida que o projeto aumenta em tamanho e em complexidade, o valor de um bom código também aumenta
- sempre revisite o código e aplique *tidyings* (mini refatoramente)
- *craftmanship* - orgulho em escrever um código durável
- Degradação de dados(bit-rot) – necessidade de atualizar um código que não é usado há algum tempo
- Dívida tech – trabalho adiado. Ocorre quando o código tem que ser escrito rapidamente m vez de adequadamente

Adaptação de requisitos

- Única constante: projetos sempre mudam!
- À medida que o projeto aumenta e é dividido em diversos scripts ou notebooks que dependem uns dos outros, pode ficar mais complexo fazer mudanças
- Fica mais fácil entender o código de outra pessoa se estiver adequadamente documentado e fácil de ler

Bom código: simplicidade

- “Complexidade é qualquer coisa relacionada à estrutura de um sistema que dificulta a compreensão e modificação de um sistema” J.Ousterhout.
- Complexidade imprevista(ou acidental) é quando não temos certeza de qual função dentro do código precisa ser alterada para conseguirmos realizar determinada ação
- Abordagens:
  - Evitar repetições
  - Manter a consição (enxuto)
  - Modularizar (quebrar em partes)

Princípio DRY

- A informação não deve ser repetida
- Don’t repeat yourself (não repita a si mesmo)
- Duplicidade aumenta a chance de bugs
- Código extenso exige mais tempo de leitura

Prática: DRY

Atividade1: digitar os 2 códigos e criar um notebook explicando o que faz cada bloco

```
[1] import pandas as pd
```

```
[2] df = pd.read_csv("sdg_literacy_rate.csv")
    df = df.drop(["Series Name", "Series Code", "Country Code"], axis=1)
    df = df.set_index("Country Name").transpose()
    df.head()
```

```
[ ] df2 = pd.read_csv("sdg_electricity_data.csv")
    df2 = df2.drop(["Series Name", "Series Code", "Country Code"], axis=1)
    df2 = df2.set_index("Country Name").transpose()
    df3 = pd.read_csv("sdg_urban_population.csv")
    df3.head()
```

```
    df3 = df3.drop(["Series Name", "Series Code", "Country Code"], axis=1)
    df3 = df3.set_index("Country Name").transpose()
    df3.head()
```

```
[1] import pandas as pd
```

```
[2] def process_sdg_data(csv_file, columns_to_drop):
        df = pd.read_csv(csv_file)
        df = df.drop(columns_to_drop, axis=1)
        df = df.set_index("Country Name").transpose()
        return df
```

```
[5] df = process_sdg_data("sdg_literacy_rate.csv", ["Series Name", "Series Code", "Country Code"])
    df2 = process_sdg_data("sdg_electricity_data.csv", ["Series Name", "Series Code", "Country Code"])
    df3 = process_sdg_data("sdg_urban_population.csv", ["Series Name", "Series Code", "Country Code"])
```

Evite código verboso

- Simplificar o código para ter menos linhas
- Conciso (enxuto) porém legível
- Usar funções built-in ao invés de criar as suas
- Evite variáveis temporárias desnecessárias
- Evite repetições (eu é que estou sendo repetitivo,não?)

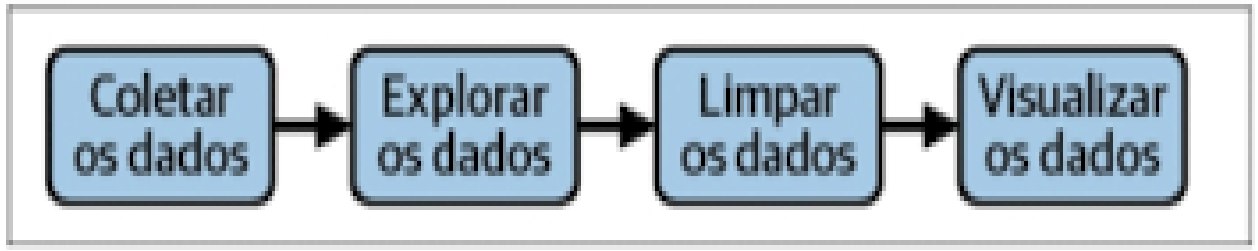
```
i = float(i)
Image_vector.append(i/255.0)
```

```
Image_vector.append(float(i)/255)
```

Modularidade

- Dividir para conquistar
- Facilita a leitura e a localização da origem do problema
- Simplifica a reutilização
- Mais fácil de testar

Modularidade



- Dividir em etapas
- Arquivos notebooks
- No final de cada um pode-se salvar um arquivo que é a entrada do próximo notebook
- Uma função extrai os dados e passa para outra limpar

```
Def load_data(csv_file):
    pass # Código será implementado depois
Def clean_data( input_data, max_lenght):
    pass
Def plot_data(clean_data, x_axis_limit, line_width)
    pass
```

Legibilidade

- É importante que outras pessoas também possam usar o seu código
- Adotar padrões e convenções
  - Bons nomes
  - Remover código não utilizado
  - documentar

Padrões e convenções

- PEP8 (python enhancement proposal 8)
- Propostas de melhorias para Python 8 (2001)
  - https://peps.python.org/pep-0008/

- Guias de estilos complementam o PEP8
  - https://google.github.io/styleguide/pyguide.html

Padrões e convenções

```
#correto
spam(ham[1], {eggs:2})

#errado
span( ham[ 1 ], { eggs: 2 } )
```

Usando ferramentas para padronizar código:
https://www.youtube.com/watch?v=j1MbEYhYj_Y

Padrões e convenções: nomes

- Nome de funções, variáveis, projetos e ferramentas

```
import pandas as p
x = p.read_csv(f, index_col=0)
```

OU

```
import pandas as pd
df = pd.read_csv(input_file, index_col=0)
```

Limpando código

- Depois de testar, remover o código que foi comentado e as chamadas desnecessárias à função print() usada para depurar o código
- Um código sujo pode ser copiado e adaptado em outros projetos!
- Teoria das janelas quebradas
- Estabelecer um alto padrão em projetos incentiva todos da equipe a escrevem bons códigos
- Refatoração

Documentação

- Ajuda outras pessoas a ler seu código
- Múltiplos níveis de detalhes:
  - Comentários simples em linhas
  - *Docstrings* que explicam uma função
  - Read.me
  - tutoriais

```python
def soma(a, b):
    """
    Retorna a soma de dois números.

    Parâmetros:
    a (int ou float): Primeiro número.
    b (int ou float): Segundo número.

    Retorna:
    int ou float: O resultado da soma de a + b.
    """
    return a + b

# Podemos acessar a docstring com:
print(soma.__doc__)
```

Desempenho

- Bons códigos precisam ser eficientes
- Métricas:
  - Tempo de execução
  - Uso de memória
- Escolher a melhor estrutura de dados
- Qual parte do código demora mais?

Robustez

- Você deve ser capaz de rodar seu código do início ao fim sem que ele falhe
- Deve ser capaz de responder com elegância se as entradas mudarem repentinamente
- Em vez de gerar um erro inesperado que pode causar falha de um sistema maior, seu código deve ser projetado para responder às mudanças
- Tratamento adequado de erros
- Registro de logs
- Bons testes

Erros e registro de logs

- Se falta metade das linhas esperadas de dados em um arquivo CSV, você deseja que seu código retorne um erro ou continue processando somente com esta metade?
- Se o erro for tratado, é importante registrar essa ocorrência

Teste de código

- Fundamentais para um software robusto
- Teste de usuário
- Teste automatizado
- Teste unitário
- Testes de ponta a ponta

Resumo

- Simplicidade
- Modularidade
- Legibilidade
- Desempenho
- Robustez