FIAP – FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

AUGUSTO DOUGLAS NOGUEIRA DE MENDONÇA - RM558371
GABRIEL VASQUEZ QUEIROZ DA SILVA – RM557056
GUILHERME ARAUJO DE CARVALHO – RM558926
GUSTAVO OLIVEIRA RIBEIRO – RM559163

CHALLENGE – DYNAMIC PROGRAMMING

SÃO PAULO
2025

SUMÁRIO

INTRODUÇÃO

A crescente digitalização dos serviços de saúde tem impulsionado a criação de sistemas computacionais que auxiliam no controle, análise e armazenamento de informações médicas. No contexto dos exames laboratoriais, o acompanhamento organizado dos resultados é essencial para diagnósticos precisos e históricos clínicos bem documentados. No entanto, muitos processos ainda são feitos de forma fragmentada ou manual, o que pode comprometer a eficiência no atendimento e a qualidade das informações armazenadas.

Diante dessa realidade, este projeto propõe o desenvolvimento de um sistema de monitoramento de exames clínicos, com foco na organização dos dados de pacientes, cadastro estruturado de exames por tipo e data, e análise automática com base em faixas de referência da literatura médica. A aplicação foi construída com interface interativa via terminal, utilizando princípios de organização modular, validação de dados e estruturação por dicionários e classes Python, de modo a garantir clareza e escalabilidade.

O sistema também contempla funcionalidades como visualização detalhada de exames, exclusão de registros e geração de diagnósticos simplificados, proporcionando ao usuário uma experiência intuitiva e confiável. A estrutura de dados foi projetada com base em hipóteses clínicas realistas, e o código foi desenvolvido de maneira a permitir futuras expansões, como persistência de dados e integração com interfaces gráficas ou bancos de dados relacionais.

1 DOCUMENTO DE ENVOLTÓRIA — SISTEMA DE MONITORAMENTO DE EXAMES CLÍNICOS

1.1 Objetivo Do Sistema

O objetivo deste sistema é permitir o cadastro, controle e análise de exames clínicos de diferentes pacientes, utilizando o conceito de uma "agenda médica digital". Ele permite que o usuário cadastre pacientes, registre exames realizados com suas respectivas datas e visualize análises automáticas com base em faixas de referência clínica.

1.2 Estrutura Do Sistema

O sistema está implementado em Python com uso de:

- @dataclass: para modelar os dados de Usuario e Exame.
- Estrutura de dicionários aninhados para organizar os dados: agenda[cpf] -> Usuario.exames[tipo_exame][data] -> Exame.
- Modularização com funções bem definidas e uso extensivo de DocStrings para documentação interna.

1.3 Funcionalidades Principais

1.3.1 Cadastro de Usuários (cadastrar_usuario)

- Solicita CPF, nome, idade e gênero.
- Garante CPF válido (11 dígitos numéricos).
- Armazena dados na estrutura agenda.

1.3.2 Cadastro de Exames (cadastrar_exame)

- Solicita CPF e verifica existência do paciente.
- Permite escolha entre 4 tipos de exames:
  - Hemograma Completo
  - Colesterol
  - Glicemia
  - Hormônios
- Coleta dados específicos de cada exame.
- Valida a data do exame e previne duplicidade (com confirmação para sobrescrever).
- Analisa os resultados com base em faixas clínicas e armazena a análise automaticamente.

1.3.3 Visualizar Todos os Exames (visualizar_exames)

- Exibe todos os exames cadastrados de um paciente, organizados por tipo e data.
- Mostra os valores inseridos e a análise gerada.

1.3.4 Visualizar Exame Específico (visualizar_exame_especifico)

- Mostra os tipos de exames disponíveis com as datas cadastradas.
- Permite escolher um tipo e uma data para exibir os detalhes.
- Mostra os dados do exame e a análise, indicando se cada item está normal, acima ou abaixo.

1.3.5 Apagar Usuário (apagar_usuario)

- Remove um paciente (e todos os seus exames) com base no CPF.

1.3.6 Menu Interativo (menu)

- Interface baseada em terminal com emojis, validações e experiência de navegação simplificada.

1.4 Validações Implementadas

- CPF deve conter exatamente 11 dígitos numéricos.
- Datas devem seguir o padrão dd/mm/aaaa.
- Gênero limitado a M ou F.
- Entradas numéricas são obrigatórias e tratadas com tratamento de exceção.

1.5 Análise Automática dos Exames

Cada exame é analisado conforme padrões clínicos:

- Hemograma: Glóbulos vermelhos, hemoglobina, leucócitos e plaquetas com faixas específicas por gênero.

- Colesterol: Limites máximos/mínimos para colesterol total, HDL e LDL.
- Glicemia: Classificação entre normal, pré-diabetes e possível diabetes.
- Hormônios: Faixas ideais para TSH e T4 livre.

A análise é armazenada no próprio objeto Exame e exibida quando necessário.

1.6 Tecnologias Utilizadas

- Linguagem: Python 3
- Conceitos:
  - Orientação a objetos com dataclass.
  - Estruturas de dados (dicts, listas).
  - Manipulação de entrada/saída no terminal.
  - Validação de dados.
  - Estrutura de menu com match-case (Python 3.10+).
  - Emojis para tornar o CLI mais amigável.

1.7 Considerações Finais

O código está pronto para execução via terminal e pode ser estendido para:

- Armazenamento em arquivo JSON para persistência de dados.
- Integração com banco de dados.
- Interface gráfica (Tkinter, PyQt, Flet).
- Exportação de exames em PDF ou .csv.

2. ANÁLISE DE ALGORITMOS E NOTAÇÃO O-GRANDE

2.1 `validar_cpf(cpf: str) -> bool`

- Descrição: Verifica se o CPF contém 11 dígitos numéricos.

- Complexidade de Tempo: O(n), onde n é o comprimento da string cpf. A função verifica cada caractere da string.

2.2 `validar_data(data: str) -> bool`

- Descrição: Valida se a data está no formato dd/mm/aaaa.
- Complexidade de Tempo: O(n), onde n é o comprimento da string data. A função utiliza strptime, que percorre a string.

2.3 `input_float(msg: str) -> float`

- Descrição: Força a entrada de um número decimal válido.
- Complexidade de Tempo: O(1) no pior caso, pois a função continua a solicitar a entrada até que um valor válido seja fornecido, mas cada tentativa de entrada é uma operação constante.

2.4 `input_int(msg: str) -> int`

- Descrição: Força a entrada de um número inteiro válido.
- Complexidade de Tempo: O(1) no pior caso, semelhante à função anterior.

2.5 `cadastrar_usuario()`

- Descrição: Cadastra um novo paciente.
- Complexidade de Tempo: O(1) para operações de entrada e validação, mas O(n) para a verificação do CPF e O(m) para a verificação do nome, onde n e m são os comprimentos do CPF e do nome, respectivamente.

2.6 `cadastrar_exame()`

- Descrição: Permite cadastrar um exame clínico para um paciente existente.
- Complexidade de Tempo: O(1) para a maioria das operações, mas O(n) para a coleta de resultados, onde n é o número de campos a serem preenchidos. A análise dos

resultados também pode ser O(k), onde k é o número de resultados a serem analisados.

2.7 `coletar_resultados_exame(tipo: str, genero: str) -> dict`

- Descrição: Coleta os resultados do exame com base no tipo e gênero.
- Complexidade de Tempo: O(1) para a coleta de cada resultado, mas O(n) no total, onde n é o número de campos a serem preenchidos.

2.8 `analisar_exame(tipo: str, resultados: dict, genero: str) -> list[str]`

- Descrição: Chama a função de análise específica para o exame.
- Complexidade de Tempo: O(1) para a chamada da função, mas O(n) para a análise, onde n é o número de resultados a serem analisados.

2.9 `visualizar_exames()`

- Descrição: Mostra todos os exames de um paciente por CPF.
- Complexidade de Tempo: O(m) onde m é o número total de exames cadastrados para o paciente, pois a função itera sobre todos os exames.

2.10 `visualizar_exame_especifico()`

- Descrição: Permite selecionar um tipo de exame e data para visualizar o resultado específico.
- Complexidade de Tempo: O(m) onde m é o número de exames do tipo selecionado, pois a função itera sobre as datas disponíveis.

2.11 `apagar_usuario()`

- Descrição: Remove um usuário e todos os seus exames com base no CPF.
- Complexidade de Tempo: O(1) para a remoção do usuário do dicionário.

3 EXPLICAÇÃO DAS HIPÓTESES E DADOS CONSIDERADOS PARA DESENVOLVER A SOLUÇÃO

3.1 Hipóteses sobre os Usuários

- Familiaridade com Tecnologia: Supomos que os usuários, como profissionais de saúde e pacientes, têm um conhecimento básico de tecnologia e podem usar uma interface de linha de comando.
- Necessidade de Acesso Rápido: Acreditamos que os usuários precisam acessar informações sobre exames de forma rápida e eficiente.

3.2 Hipóteses sobre os Dados

- Formato Padrão: Assumimos que os dados de entrada, como CPF e datas, seguirão formatos padrão, como CPF com 11 dígitos e datas no formato dd/mm/aaaa.
- Validação Essencial: Consideramos que a validação de dados é crucial para garantir a integridade das informações, implementando verificações rigorosas.

3.3 Dados Considerados

- Estrutura de Dados: Utilizamos dicionários para armazenar informações sobre pacientes e exames, permitindo acesso rápido e operações eficientes.
- Tipos de Exames: Escolhemos exames comuns e relevantes, como Hemograma Completo e Glicemia, baseando-nos em sua frequência na prática clínica.
- Resultados e Análises: Definimos faixas de referência e critérios de análise com base em diretrizes médicas, garantindo relevância nas análises.

3.4 Hipóteses sobre o Ambiente de Execução

- Execução Local: Supomos que o sistema seria executado em um ambiente local, como um computador pessoal, sem necessidade de infraestrutura complexa.
- Persistência de Dados: Inicialmente, consideramos que os dados seriam armazenados em memória, com a possibilidade de persistência em banco de dados para versões futuras.

3.5 Limitações e Considerações Finais

- Escalabilidade: A solução foi projetada para um número moderado de usuários e exames, necessitando de otimizações para escalabilidade em larga escala.
- Feedback dos Usuários: Acreditamos que o feedback contínuo é essencial para a evolução do sistema, permitindo melhorias e novas funcionalidades.

CONCLUSÃO

Este sistema de monitoramento de exames clínicos foi desenvolvido com um objetivo claro: criar uma solução prática e acessível que facilite o gerenciamento de pacientes, o armazenamento de exames e a análise automática de resultados com base em faixas clínicas de referência. Para garantir uma implementação eficiente, um documento técnico serviu como guia, detalhando a estrutura do sistema e explicando o uso de *dataclass* para modelagem de dados e dicionários aninhados para organização. Além disso, ele apresenta as funcionalidades principais e os fluxos operacionais.

As escolhas de design foram feitas com base em aspectos realistas, como a identificação dos pacientes pelo CPF, a utilização de dados clínicos confiáveis e uma interface intuitiva no terminal, complementada por emojis para uma experiência mais visual. Os exames armazenados são analisados cuidadosamente levando em conta variáveis como gênero e limites clínicos, garantindo que os usuários tenham acesso a informações claras e relevantes. As validações aplicadas asseguram que os dados inseridos sejam coerentes, seguros e fáceis de navegar, mesmo para quem não tem experiência técnica.

Do ponto de vista de desempenho, o sistema opera de forma eficiente, com funções de tempo constante ou linear, o que torna seu uso escalável na prática. A estrutura modular do código, aliada ao uso estratégico de dicionários, facilita a manutenção e expansão do sistema no futuro. Dessa forma, este projeto se destaca como uma solução bem planejada para o controle de exames clínicos, equilibrando simplicidade, clareza e precisão.

REFERÊNCIAS BIBLIOGRÁFICAS

Artigo científico: SILVA, João; SOUZA, Maria. *Valores de referência para exames laboratoriais de colesterol, hemoglobina glicada e creatinina da população adulta brasileira*. Revista Brasileira de Epidemiologia, v. 22, Suppl. 02, 2019. Disponível em: SciELO. Acesso em: 15 jun. 2025.

Site especializado: PINHEIRO, Pedro. *Valores de referência dos principais exames laboratoriais*. MD.Saúde, 2025. Disponível em: MD.Saúde. Acesso em: 15 jun. 2025.

Documento acadêmico: PUC-RIO. *Referências bibliográficas sobre exames de sangue e hemoterapia*. Rio de Janeiro: PUC-Rio, 2025. Disponível em: PUC-Rio. Acesso em: 15 jun. 2025.