Relatório Problema 3: Bubble Tea

Gustavo Silva Ribeiro

Universidade Estadual de Feira de Santana (UEFS)

Av. Transnordestina, s/n, Novo Horizonte, Feira de Santana – BA, Brasil – 44036-900

tavoribeiro007@gmail.com

Resumo. *Este relatório apresenta o desenvolvimento de um sistema em Python para gerenciamento de pedidos personalizados de Bubble Tea, fruto de uma proposta da cooperativa MPFSA em parceria com a UEFS. A aplicação permite ao cliente montar sua bebida escolhendo base e complementos, com descontos por categoria e sistema de cashback. O sistema utiliza conceitos de Programação Orientada a Objetos, modularização e persistência de dados com JSON.*

1. Introdução

A cooperativa MPFSA (Mulheres Poderosas de Feira de Santana), em parceria com acadêmicos da UEFS (Universidade Estadual de Feira de Santana), decidiram propor a criação de uma nova empresa que oferecesse algo inovador e culturalmente inédito na cidade. Nesse contexto, surgiu a ideia de introduzir o Bubble Tea [1] (também conhecido como chá boba, chá de bolhas ou chá perolado). Trata-se de uma bebida tradicional à base de chá, geralmente preto ou verde. A bebida é caracterizada por incluir bolinhas de tapioca (também chamadas de boba ou pérolas de tapioca), que são bolinhas gelatinosas que estouram na boca ao serem mastigadas. Criada em Taiwan por volta de 1980, rapidamente se espalhou pelo mundo, tornando-se bastante popular.

Com isso, foi estabelecido como objetivo o desenvolvimento de um sistema em linguagem Python (versão 3.13, 64-bit) capaz de gerenciar pedidos personalizados da bebida. O sistema deveria utilizar conceitos fundamentais como programação orientada a objetos, persistência de dados com a biblioteca JSON [2], e modularização entre as partes do código. Para isso, dois módulos foram criados: um dedicado à lógica de controle e interface com o usuário (programa_principal.py) e outro responsável pelas regras de negócio e estruturas de dados (dados.py). Os dados dos clientes e seus pedidos são armazenados em arquivos .txt no formato JSON, promovendo praticidade na leitura e escrita das informações.

2. Metodologia

A construção do sistema foi realizada por meio de sessões tutoriais, nas quais foram discutidos métodos, soluções e definições de funcionalidades. Dentre as ideias levantadas, destacam-se: a criação de um sistema de identificação do cliente por ID (com possibilidade futura de extensão para CPF ou e-mail), uso de histórico de pedidos acessível ao final de cada transação, e a adoção da biblioteca json para armazenar os dados de forma estruturada e segura.

2.1. Requisitos e Funcionalidades

O sistema deverá permitir que cada pedido contenha apenas uma base, podendo ser uma dentre as seguintes opções: leite, maracujá, rosa ou manga. Além disso, o cliente poderá

escolher livremente os complementos do seu pedido, podendo selecionar quantos quiser, inclusive nenhum. Entre as opções disponíveis estão: boba, lichia, geleia, taro e chia. Não haverá restrição quanto à quantidade de complementos escolhidos. Clientes que forem estudantes da UEFS terão direito a um desconto de 25% sobre o valor total do pedido. Já os professores e funcionários da UEFS receberão um desconto fixo de R$1,00. Independentemente da categoria do cliente, todos receberão 10% de cashback sobre o valor total do pedido, valor esse que poderá ser utilizado em compras futuras.

Antes da finalização de cada pedido, o sistema deverá exibir o valor total do pedido, o saldo de cashback acumulado, e, caso esse saldo seja maior que zero, o cliente deverá ser consultado se deseja utilizá-lo como forma de pagamento parcial. O resumo do pedido deverá apresentar todos os valores monetários com duas casas decimais, garantindo a clareza das informações. O sistema deverá permitir a realização de apenas um pedido por vez.

Para a implementação, será necessário identificar corretamente as entradas e saídas envolvidas no processo. Deve-se aplicar o conceito de orientação a objetos, garantindo também a persistência dos dados para que as informações possam ser armazenadas e recuperadas posteriormente. A estrutura do código precisa estar bem modularizada, de forma que cada parte cumpra sua função específica. Também é fundamental a utilização de diferentes tipos de dados, escolhendo os mais adequados conforme o contexto e a necessidade de cada funcionalidade.

2.2. Descrição do algoritmo

O sistema desenvolvido tem como objetivo gerenciar pedidos personalizados de Bubble Tea, incluindo o cadastro de clientes, aplicação de descontos por categoria, cálculo de valores com cashback e armazenamento de históricos de pedidos. A lógica do sistema é dividida em dois principais módulos:

- programa_principal.py: Controla o fluxo principal do programa (interface com o usuário e decisões);

- dados.py: Contém as classes e dados essenciais, como clientes, valores das bases e complementos.

2.2.1. Visão Geral do Sistema

O sistema desenvolvido tem como propósito gerenciar pedidos de bebidas personalizadas, oferecendo funcionalidades como seleção de base e complementos, cálculo automático de valores com aplicação de descontos e cashback, além de persistência e recuperação dos dados de clientes e pedidos. A aplicação é construída em Python, utilizando conceitos fundamentais da Programação Orientada a Objetos (POO), persistência de dados com arquivos JSON [5], e modularização por meio da separação entre a lógica de domínio (dados.py) e a lógica de controle e interação com o usuário (programa_principal.py).

2.2.1.1. Programação Orientada a Objetos (POO)

A estrutura do sistema foi concebida a partir da Programação Orientada a Objetos (POO), permitindo organizar o código em torno de entidades com responsabilidades bem definidas. Duas classes principais foram desenvolvidas: Cliente e Pedido. A classe Cliente encapsula informações e comportamentos relacionados a cada usuário do sistema, como identificação,

tipo e saldo de cashback. Já a classe Pedido representa a transação realizada, contendo informações como base escolhida, complementos adicionados, valor total e aplicação de descontos. Essa abordagem orientada a objetos promove reutilização de código, clareza na estrutura e facilidade na manutenção do sistema [6].

2.2.1.2. Funções do Python Utilizadas

O programa faz uso de diversas funções nativas da linguagem Python, como input() [4], utilizada para captura de dados via terminal, print() para exibição de informações, e open() [3] para leitura e escrita de arquivos. A estrutura de controle while é empregada para manter o menu principal em execução até que o usuário decida encerrar. Funções auxiliares foram desenvolvidas para modularizar trechos recorrentes, como seleção de bases e complementos, além de tratamento de exceções com try e except para evitar falhas por entradas inválidas.

2.2.1.3. Persistência de Dados

A persistência de dados foi garantida por meio da biblioteca json, que permite serializar objetos Python em strings JSON e armazená-las em arquivos .txt. As informações dos clientes são salvas no arquivo clientes.txt, enquanto os pedidos finalizados são armazenados em pedidos.txt. Esse processo garante que os dados sejam mantidos entre diferentes execuções do programa, possibilitando histórico e recuperação de informações anteriores. Cada linha dos arquivos representa um objeto JSON individual, facilitando a leitura linha a linha.

2.2.1.4. Estratégias de Persistência Empregadas

Para manter os dados de forma organizada e acessível, o sistema utiliza a estratégia de persistência conhecida como JSON Lines, onde cada linha de um arquivo .txt representa um objeto JSON. Ao cadastrar um novo cliente ou registrar um pedido, os dados são convertidos para dicionários Python e gravados com json.dump(). Para leitura, a função json.loads() é usada para reconstruir os dados em estruturas Python. Essa abordagem permite operações eficientes de busca, leitura e atualização de registros, além de facilitar a manipulação dos dados sem necessidade de um banco de dados relacional. A função atualizar_cliente(), por exemplo, percorre o arquivo todo, edita a linha correspondente ao cliente e reescreve o arquivo com os dados atualizados, demonstrando controle sobre persistência e atualização sequencial.

2.2.2.1. Módulo programa_principal.py

O módulo programa_principal.py é o ponto de entrada da aplicação e tem como principal responsabilidade montar o fluxo de execução do programa, controlando a interação com o usuário por meio de um menu textual e gerenciando operações específicas ao módulo dados.py. Esse módulo atua como camada de controle, isolando a lógica de apresentação da lógica de negócio, o que está de acordo com o princípio de modularização funcional.

A estrutura principal do módulo é composta por um laço de repetição while. A interação com o usuário é feita por meio da função input(), que captura as escolhas numéricas e textos informados durante a execução. As três opções que implementam o menu principal são:

1. Criar um novo pedido

2. Exibir pedidos anteriores
3. Encerrar o programa

Cada opção é mapeada para uma funcionalidade específica, controlando a navegação do usuário de maneira clara e sequencial.

A primeira etapa na criação de um pedido é a identificação do cliente. O sistema solicita o ID do cliente (obrigatoriamente com 4 dígitos numéricos) e invoca a função verificar_id() da classe Cliente, localizada no módulo dados.py. Se o cliente já estiver cadastrado, seus dados são recuperados e reutilizados. Caso contrário, um novo cadastro é realizado por meio da função criar_cliente().

Uma vez validado o cliente, o sistema conduz a montagem do pedido. A base da bebida é escolhida entre quatro opções fixas, seguida da seleção de múltiplos complementos, se desejado. O uso de lista permite registrar os índices dos complementos escolhidos, garantindo flexibilidade sem impor um número fixo de itens. Essa etapa é acompanhada de validação de entrada com tratamento de exceções, assegurando a estabilidade da aplicação mesmo em cenários de entrada inválida.

Na consulta e aplicação de cashback, se o cliente possuir cashback acumulado, o sistema exibe o valor disponível e solicita confirmação sobre seu uso no pedido atual. A resposta do usuário determina se o atributo cashback do objeto Cliente será descontado do total do pedido. Essa manipulação de atributos reforça o uso de estado interno dos objetos e encapsulamento.

Após coleta dos dados, é instanciado um objeto da classe Pedido, passando-se os dados do cliente, a base e os complementos como parâmetros. O método calcular_total() é então chamado para processar o valor final do pedido, aplicando descontos específicos conforme o tipo de cliente (estudante, professor/funcionário ou comum) e calculando o cashback gerado. O método resumo_pedido() exibe um resumo do pedido formatado, além de registrar os dados em arquivo. A responsabilidade pela lógica de negócio, como regras de desconto, cálculo de cashback e persistência de dados, é totalmente delegada ao módulo dados.py.

Por fim, a opção de visualização permite que o usuário consulte todos os pedidos registrados ou apenas os de um cliente específico. A leitura é feita diretamente do arquivo pedidos.txt, utilizando a biblioteca JSON para carregar e exibir os dados de forma estruturada.

2.2.2.2. Módulo dados.py

O módulo dados.py é o núcleo lógico da aplicação. Nele estão definidos os elementos centrais que estruturam o sistema: o cliente e o pedido. Aqui se concentram tanto as regras de negócio como descontos, cashback e cálculos de valores, quanto os mecanismos de persistência de dados. Todo esse controle é implementado usando conceitos sólidos de programação orientada a objetos, além de boas práticas para a modularização e uso adequado de tipos de dados compostos.

No início do módulo, são declaradas duas estruturas de dados fundamentais: os dicionários bases e complementos. Esses dicionários armazenam os nomes dos itens disponíveis (base e adicionais da bebida) como chaves e seus respectivos preços como

valores. Posteriormente, listas auxiliares são extraídas a partir desses dicionários, como lista_bases e itens_b (valores das bases), bem como lista_complementos e itens_c (valores dos complementos). Essa abordagem permite que o programa trabalhe internamente com índices ao invés de strings.

A primeira entidade definida é a classe Cliente. Ela encapsula os atributos ID, nome, tipo e cashback, todos definidos no construtor __init__. Essa classe oferece três funções fundamentais: criar_cliente(), verificar_id() e atualizar_cliente(). O método criar_cliente() tem como objetivo realizar a persistência do objeto cliente, armazenando seus dados no arquivo clientes.txt. A operação é feita utilizando a biblioteca JSON, com os dados sendo serializados em formato JSON por linha (JSON lines), o que facilita a leitura e a escrita.

A função verificar_id() é usada para procurar um cliente pelo seu ID. Ela percorre o arquivo clientes.txt linha por linha e, se encontrar um cliente com o ID correspondente, cria e retorna um objeto Cliente com os dados carregados. Caso contrário, retorna False. Essa função é crucial para evitar cadastros duplicados e permite o reaproveitamento de dados de clientes já existentes.

O método atualizar_cliente() serve para atualizar o cashback do cliente após um novo pedido. Ele lê o conteúdo do arquivo, altera os dados do cliente correspondente e grava todo o conteúdo.

A classe Pedido representa um pedido feito por um cliente. Ao ser instanciada, recebe o cliente, a base escolhida, os complementos e se o cliente autorizou o uso do cashback. Os métodos dessa classe cuidam de todo o processamento envolvido na finalização do pedido.

O método calcular_total() faz o cálculo do valor final. Primeiro, soma o valor da base e dos complementos escolhidos. Depois, aplica os descontos: 25% para estudantes ou R$1,00 para professores/funcionários. Se o cliente tiver cashback e optar por usá-lo, o valor é descontado do total, e o saldo é zerado. Ao final, o cliente recebe 10% do valor total como novo cashback, que é acumulado para o próximo pedido.

O método resumo_pedido() exibe todas as informações relevantes de forma organizada: base, complementos, tipo de cliente, valor final, valor do cashback e o novo saldo. Além disso, todos os dados são gravados em pedidos.txt em formato JSON, garantindo o histórico de pedidos realizados.

2.3. Ordem de Codificação

Seguindo as metas ordenadas nas sessões tutoriais, a estrutura ocorreu do seguinte modelo até sua finalização:

- Estruturar menus e entradas;
- Criar as classes, atributos e métodos;
- Modularizar o código, criando o progama_principal.py e dados.py;
- Usar JSON para persistência de dados.

2.4. Ferramentas

O projeto foi desenvolvido com a linguagem Python (versão 3.13), utilizando o editor de código Visual Studio Code (versão 1.100.3) e executado no sistema operacional Windows 11 (versão 24H2).

3. Resultados e Discussões

Nesta seção, apresenta-se uma análise detalhada da solução desenvolvida, destacando-se suas principais funcionalidades e o funcionamento geral do sistema. A aplicação foi implementada com foco na usabilidade, integridade dos dados e fidelidade aos requisitos propostos. A seguir, são descritas as instruções de uso do programa, bem como os resultados obtidos durante os testes, incluindo possíveis falhas e limitações identificadas.

3.1. Manual de Uso

O programa foi desenvolvido com uma interface baseada em texto, que conduz o usuário de forma sequencial e interativa por meio de menus. A navegação ocorre por meio de entradas numéricas simples, facilitando seu uso.

3.1.1. Requisitos

Para que o programa seja executado corretamente, é necessário que o ambiente de desenvolvimento atenda aos seguintes requisitos mínimos:

- Python: Versão 3.13 (64-bit) ou superior instalada na máquina;
- Editor de código: Preferencialmente Visual Studio Code (ou qualquer outro que permita executar scripts Python);
- Sistema Operacional: Compatível com Windows 11 ou qualquer sistema que suporte Python 3;
- Permissão de escrita: A pasta onde o programa está salvo deve permitir leitura e escrita de arquivos .txt.

Além disso, os arquivos programa_principal.py e dados.py devem estar no mesmo diretório, já que o primeiro importa diretamente classes e variáveis do segundo.

3.1.2. Execução do Programa

Ao iniciar, o sistema exibe o menu principal, que contém três opções: Novo pedido (1), Mostrar pedidos (2) e Sair (3) conforme a Figura 1.

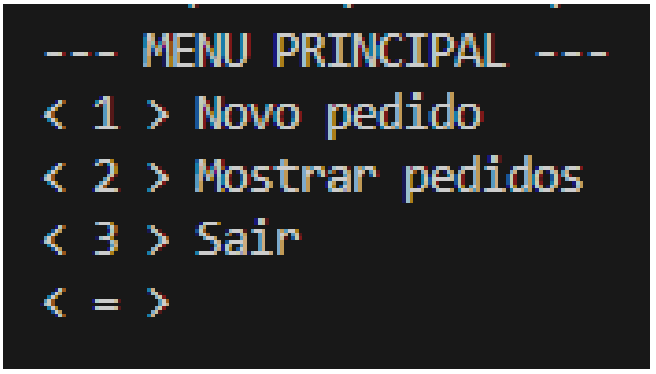

Figura 1. Menu principal do programa

3.1.2.1. Novo pedido

Ao selecionar a opção 1, o sistema inicia o processo de identificação ou cadastro do cliente, solicitando um ID de 4 dígitos. Se o ID já estiver presente no arquivo clientes.txt, os dados são recuperados. Caso contrário, um novo cadastro é solicitado com nome e tipo (Estudante, Professor/Funcionário ou Comum). A seguir, o cliente escolhe uma base de bebida entre as quatro disponíveis. Depois, é oferecida a possibilidade de adicionar complementos, podendo escolher quantos quiser ou nenhum. Após a montagem do pedido, caso o cliente possua cashback acumulado, o sistema oferece a opção de utilizá-lo como desconto.

3.1.2.2. Mostrar pedido

Permite visualizar o histórico de pedidos (Figura 2), seja por cliente específico (buscando pelo ID) ou exibindo todos os pedidos registrados.

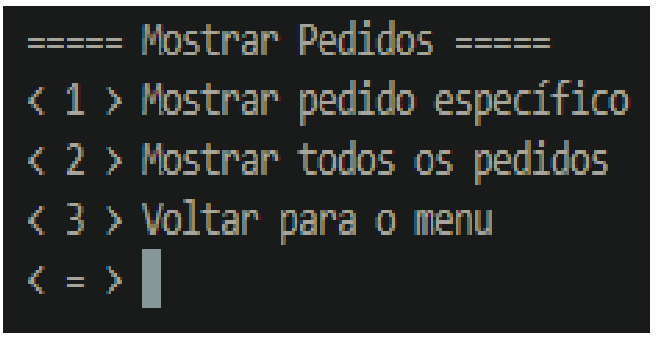

Figura 2. Escolha da exibição do pedido

3.1.2.3. Sair

O programa é finalizado.

3.2. Testes e Erros

Durante a fase de testes do sistema, foram identificados alguns desafios e erros comuns que surgiram ao longo do desenvolvimento. A seguir, destacam-se os principais pontos enfrentados, bem como suas causas e soluções aplicadas:

Um dos primeiros cuidados necessários foi garantir a validação correta das entradas numéricas durante a seleção de bases e complementos. Como o sistema utiliza os índices dessas opções para acessar valores nos dicionários bases e complementos, era essencial evitar acessos inválidos ou fora do intervalo. Inicialmente, houve casos em que entradas não numéricas ou índices incorretos resultavam em erros de execução. Isso foi resolvido com a implementação do try, except e verificações explícitas dos valores aceitos.

Outro erro frequente foi tentar utilizar os índices diretamente nos dicionários, como se fossem chaves. Isso ocorria, por exemplo, ao tentar acessar bases no índice 1 ou complementos no índice 2, o que naturalmente gerava exceções, já que os dicionários possuem strings como chave ('leite', 'boba', etc.), e não inteiros. A solução adotada foi mapear os índices para listas auxiliares (lista_bases, lista_complementos, itens_b, itens_c), que atuam como ponte entre a

entrada do usuário e os dados nos dicionários originais. Essa separação lógica também facilitou a formatação e apresentação das informações no resumo do pedido.

Um erro sutil, mas que impactava diretamente o resultado do pedido, era esquecer de subtrair 1 ao salvar o número digitado pelo usuário na lista de complementos (comple). Como o menu apresenta as opções numeradas de 1 a 5, e as listas internas são indexadas a partir do zero, isso causava acesso a índices incorretos ou mesmo gerava IndexError. Esse problema foi resolvido padronizando a lógica de entrada, com a subtração do valor digitado sempre que o índice era armazenado.

Outro ponto crítico foi observado na função atualizar_cliente(), da classe Cliente. Em sua primeira versão, o c ódigo tentava ler e reescrever o arquivo clientes.txt dentro de um único bloco with open(..., "w") [3], o que causava perda de dados, já que ao abrir um arquivo em modo de escrita, seu conteúdo é imediatamente truncado. Para corrigir isso, foi necessário utilizar um bloco separado para leitura (modo 'r') antes da escrita, garantindo que as informações fossem lidas corretamente antes da substituição. Esse ajuste garantiu que apenas os dados alterados fossem atualizados e que os demais permanecessem intactos.

Apesar desses obstáculos, todos os erros foram solucionados com ajustes simples e eficazes, reforçando a importância dos testes contínuos durante o processo de desenvolvimento.

4. Conclusão

O desenvolvimento do sistema de pedidos de Bubble Tea alcançou com sucesso os objetivos propostos no início do projeto. Foram implementadas todas as funcionalidades previstas, incluindo a seleção personalizada de bebidas, aplicação de descontos conforme a categoria do cliente, geração e uso de cashback, além da persistência e recuperação de dados utilizando a biblioteca JSON. Não houve funcionalidades deixadas de lado, e o projeto foi finalizado de acordo com o acordo proposto.

4.1. Melhorias

Apesar de o sistema funcionar corretamente e atender aos requisitos propostos, algumas melhorias podem ser implementadas. A seguir, destacam-se sugestões que podem ser aplicadas em futuras versões:

- **Remoção de complementos já adicionados:** Atualmente, o usuário pode apenas adicionar complementos em sequência, sem possibilidade de revisar ou remover algum item escolhido por engano. Uma melhoria útil seria permitir que o cliente visualize a lista de complementos já selecionados e tenha a opção de remover algum antes de finalizar a escolha.
- **Melhoria na interface do terminal:** A interface atual, apesar de funcional, pode ser melhorada com elementos simples de formatação, como espaçamentos, divisórias visuais ("=", "—"), uso de cores, e mensagens mais amigáveis.

- **Limpeza de tela entre as etapas:** Para evitar acúmulo de informações no terminal, pode-se incluir uma função de "limpar tela" entre as etapas do pedido, utilizando os.system('cls') no Windows ou os.system('clear') no Linux/macOS, deixando a interface mais limpa a cada nova ação.
- **Cancelar pedido:** Permitir que um pedido em andamento possa ser cancelado e retomado ou que seja confirmado apenas ao final, evitando gravação imediata antes da revisão final pelo usuário.

5. Bibliografia

1. BUBBLE tea. *Wikipedia: The Free Encyclopedia*. Disponível em: https://en.wikipedia.org/wiki/Bubble_tea. Acesso em: 06 jun. 2025.
2. PYTHON SOFTWARE FOUNDATION. *json — JSON encoder and decoder*. *Documentação Python 3.13.4*. Disponível em: https://docs.python.org/3/library/json.html. Acesso em: 06 jun. 2025.
3. PYTHON SOFTWARE FOUNDATION. *open — Função de leitura e escrita de arquivos*. Documentação Python 3.13.4. Disponível em: https://docs.python.org/3/library/functions.html#open. Acesso em: 06 jun. 2025.
4. PYTHON SOFTWARE FOUNDATION. *input — Função de entrada de dados*. Documentação Python 3.13.4. Disponível em: https://docs.python.org/3/library/functions.html#input. Acesso em: 06 jun. 2025.
5. ARAÚJO, A. M.; COELHO, H. A.; MENEZES, P. C. *Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes*. Rio de Janeiro: Elsevier, 2020. Disponível em: https://www.elsevier.com.br/produto/introducao-a-algoritmos-e-programacao-com-python-uma-abordagem-dirigida-por-testes-34560. Acesso em: 06 jun. 2025.
6. ZANDER, A. *Python Orientado a Objetos*. São Paulo: Novatec, 2019.