Desenvolvimento de um Sistema Piloto Simples que Gerencie o Programa de Transporte Público da Cidade de TechVille

Thiago Marques Reis[1]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana – BA – Brasil

theicefrosts@gmail.com

Resumo. *Este relatório apresenta o desenvolvimento de um sistema piloto simples para o programa de transporte público da cidade de TechVille. Ele deverá ter a capacidade de gerenciar recargas, atribuir os descontos das passagens e verificar se o saldo é suficiente para embarcar no ônibus, ou seja, suficiente para comprar passagens. Além disso, o administrador do sistema deverá ter a possibilidade de solicitar um relatório com todos os dados armazenados. O programa foi produzido com a linguagem de programação Python.*

Palavras-chave: Sistema Piloto Simples. Transporte Público. Python.

1. Introdução

Inúmeros foram os desafios para a resolução desse problema, cujo objetivo é desenvolver uma solução para uma questão pública e humanitária, que é a ausência de um sistema piloto funcional e realmente eficiente, onde os usuários não terão grandes dificuldades ao utilizar o programa, que será executado na cidade de TechVille, mais especificamente na modernização do sistema de transporte público.

A motivação para resolver o problema é a de conseguir criar algo que impacte a vida de pessoas reais, que tem centenas de questões para resolver e não conseguem sequer ficar tranquilas e sem se estressar devido a complicações em locais de transporte público.

A solução do problema consiste em um projeto capaz de prover um sistema que gerencie recargas, organize os descontos das passagens de acordo com cada categoria e faça as verificações necessárias antes de confirmar alguma informação. Além disso, existe um relatório que teoricamente deve imprimir e guardar as informações que são inseridas a todo momento. Para esse projeto, dei-lhe o nome de TechVille Transportes.

Em uma breve descrição, podemos dizer que o programa recebe como entradas o valor da passagem, a opção do menu que o usuário quer executar, a sua respectiva categoria e dados como a quantidade da recarga ou de passagens. Como saídas, temos os respectivos dados inseridos e no caso do relatório do administrador, há a acumulação desses dados ao longo do tempo. A lógica é simples, há a configuração do valor da passagem, o usuário escolhe no menu entre imprimir um relatório (administrador), fazer uma recarga, comprar uma passagem e sair, todas essas funcionalidades com um submenu para cada categoria. Caso o usuário digite algo errado, o sistema retorna ao menu principal.

2. Metodologia

Neste trabalho, houveram decisões coletivas e individuais. Entre as principais decisões coletivas estão a definição prévia de variáveis bem definidas, a necessidade do tratamento de dados, o uso de um menu para organizar as funcionalidades do programa e o uso de estruturas condicionais e de repetição. Já em relação às principais decisões individuais estão a possibilidade do relatório ser impresso no menu principal e nos submenus, o fato do programa se repetir infinitamente, o uso do "try-except" para evitar que caracteres indesejados "quebrem o código" (exceção autorizada pelo tutor, mas que não interferirá na pontuação final) e o uso da função "exit" caso o usuário deseje que o programa se encerre.

2.1. Requisitos

Entre as exigências que foram impostas para o desenvolvimento do sistema piloto simples estão:

- Registrar recargas de acordo com o tipo de usuário.
- Atribuir os descontos das passagens de acordo com o tipo de usuário.
- Verificar se o saldo é suficiente para embarcar no ônibus, ou seja, suficiente para comprar passagens.
- Possibilidade de comprar passagens.
- Ao inicializar, o programa deverá permitir que o valor da passagem seja cadastrado.
- A qualquer momento, o administrador deve ter a possibilidade de solicitar um relatório ao sistema, contendo:
    1. Quantidade de recarga total e o número de recargas realizadas por cada usuário/categoria.
    2. Quantidade de passagens usadas (compradas) por cada usuário/categoria.

3. Valor total gasto com passagens por cada usuário/categoria.
4. Saldo restante a ser utilizado por cada usuário/categoria.

2.2. Descrição do Algoritmo

O programa permite definir o valor da passagem, recarregar cartões, comprar passagens para diferentes tipos de usuários (padrão, estudante/idoso, social) e imprimir os respectivos relatórios de acordo com a categoria. O programa é executado infinitamente até que o usuário escolha a opção de sair.

```
#Declaração de variáveis para a compra das passagens de acordo com o usuário.
comprar_passagem_padrao = 0
comprar_passagem_estudante_idoso = 0
comprar_passagem_social = 0

#Declaração de variáveis para recargas de acordo com o usuário.
fazer_recarga_padrao = 0
fazer_recarga_estudante_idoso = 0
fazer_recarga_social = 0

#Declaração de variável para definir o valor da passagem.
valor_da_passagem = 0

#Declaração de variáveis para definir os menus.
menu = 0
menu_0 = 0
menu_1 = 0
menu_2 = 0

#Declaração de variáveis para futura impressão da quantidade de recarga total de acordo com o usuário.
quantidade_recarga_total_padrao = 0
quantidade_recarga_total_estudante_idoso = 0
quantidade_recarga_total_social = 0
```

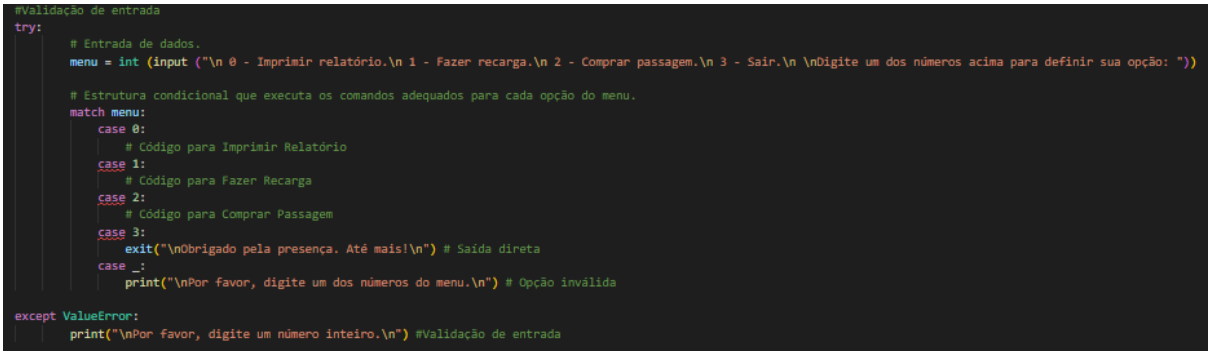
Figura 1. Inicialização de variáveis.

No início do programa, diversas variáveis são declaradas e inicializadas. Elas servem para armazenar os dados de cada tipo de usuário (saldos, totais recarregados, contadores de operações, etc.), o valor da passagem, e controlar a navegação nos menus e a execução dos loops principais. Essa imagem somente exemplifica e mostra a importância desse processo.

```
#Estrutura de repetição que repete o código completo.
while continuar_1:

    #Estrutura de repetição que encerra quanto o valor da passagem for válido.
    while continuar_2:

        #Estrutura que realiza a validação e verifica se ele é um número inteiro ou de ponto flutuante.
        try:

            #Imprime uma frase de boas-vindas.
            print("\nBem vindo à TechVille Transportes!\n")

            #Entrada de dados.
            valor_da_passagem = float (input("Digite o valor da passagem: "))

            #Estrutura condicional que verifica se o valor da passagem é maior do que 0.
            if valor_da_passagem > 0:
                print(f"\nRegistrado! Agora, a passagem custa {valor_da_passagem} créditos.\n")

                #Interrompe o loop.
                continuar_2 = False
            else:
                print("\nPor favor, digite um número maior do que 0.\n")

        except ValueError:
            print("\nPor favor, digite um número inteiro ou de ponto flutuante.\n")
```

Figura 2. Estrutura de Repetição Principal e Definição do Valor da Passagem.

O programa funciona dentro de uma estrutura de repetição principal (while continuar_1) que permite ao usuário realizar as ações seguintes. No início de cada ciclo deste loop principal, um loop secundário (while continuar_2) solicita o valor da passagem. Existe um tratamento de dados que impede o valor de ser menor ou igual a 0 e uma estrutura de validação "try-except" que evita a entrada de caracteres não numéricos.



```
#Validação de entrada
try:
        # Entrada de dados.
        menu = int (input ("\n 0 - Imprimir relatório.\n 1 - Fazer recarga.\n 2 - Comprar passagem.\n 3 - Sair.\n \nDigite um dos números acima para definir sua opção: "))

        # Estrutura condicional que executa os comandos adequados para cada opção do menu.
        match menu:
            case 0:
                # Código para Imprimir Relatório
            case 1:
                # Código para Fazer Recarga
            case 2:
                # Código para Comprar Passagem
            case 3:
                exit("\nObrigado pela presença. Até mais!\n") # Saída direta
            case _:
                print("\nPor favor, digite um dos números do menu.\n") # Opção inválida

except ValueError:
        print("\nPor favor, digite um número inteiro.\n") #Validação de entrada
```

Figura 3. Ilustração do Menu Principal.

Após a definição do valor da passagem, um menu principal é apresentado ao usuário, solicitando a escolha de uma ação (0 a 3). A entrada é verificada usando o "try-except" mais uma vez. A estrutura "match menu" direciona o fluxo do programa para a funcionalidade correspondente (imprimir relatório, fazer recarga, comprar passagem ou sair). Entradas inválidas resultam em uma mensagem de erro.

```
# Dentro do case 0 do menu principal:
try:
    menu_0 = int (input("\n 1 - Usuário padrão.\n 2 - Usuário estudante/idoso.\n 3 - Usuário social.\n ..."))
    match menu_0:
        case 1:
            # Imprime um relatório para o usuário padrão.
            print("\nRelatório do usuário padrão: ")
            print(f"\nQuantidade de recarga total = {quantidade_recarga_total_padrao} créditos.")
            print(f"Número de recargas = {numero_recargas_padrao} recargas.")
            print(f"Quantidade de passagens utilizadas = {quantidade_passagens_compradas_padrao} passagens.")
            print(f"Valor total gasto com passagens = {valor_total_gasto_passagens_padrao} créditos.")
            print(f"Saldo restante = {saldo_restante_padrao} créditos.")
        case 2:
            # Imprime um relatório para o usuário estudante/idoso.
        case 3:
            # Imprime um relatório para o usuário social.
        case _:
            print("\nPor favor, digite um dos números do menu.\n")
except ValueError:
    print("\nPor favor, digite um número inteiro.\n")
```

Figura 4. Ilustração da impressão do relatório.

Apresenta um submenu (menu_0) para selecionar o tipo de usuário. Em seguida, imprime os dados armazenados ao longo da execução do programa nas variáveis correspondentes (total recarregado, nº de recargas, passagens usadas, gasto total e saldo restante, em resumo).

```
case 1:

    #Estrutura que realiza a validação e verifica se ele é um número inteiro ou de ponto flutuante.
    try:

        #Entrada de dados.
        fazer_recarga_padrao = float (input("\nDigite quanto você quer recarregar: "))

        #Estrutura condicional que verifica se o valor da recarga é maior do que 0.
        if fazer_recarga_padrao > 0:

            #Atribuições para as variáveis (processamento de dados).
            quantidade_recarga_total_padrao += fazer_recarga_padrao
            numero_recargas_padrao += 1
            saldo_restante_padrao += quantidade_recarga_total_padrao

            print (f"\nParabéns! Você fez uma recarga de {fazer_recarga_padrao} créditos.\n")
        else:
            print ("\nPor favor, digite um número maior do que 0.\n")

    except ValueError:
        print("\nPor favor, digite um número inteiro ou de ponto flutuante.\n")
```

Figura 5. Funcionalidade para fazer recarga.

Apresenta um submenu (menu_1) para selecionar o tipo de usuário. Solicita o valor da recarga, verifica se é numérico (float) e positivo. Se a validação passar, atualiza as variáveis de quantidade de recarga total, número de recargas e saldo restante para o usuário selecionado.

```
case 1:

    #Estrutura que realiza a validação e verifica se ele é um número inteiro.
    try:

        #Entrada de dados.
        comprar_passagem_padrao = int (input("\nDigite quantas passagens você quer comprar: "))

        #Estrutura condicional que verifica se o valor da recarga é maior do que 0.
        if comprar_passagem_padrao > 0:
            #Estrutura condicional que verifica se o saldo é suficiente.
            if saldo_restante_padrao >= (comprar_passagem_padrao * valor_da_passagem):

                #Atribuições para as variáveis (processamento de dados).
                valor_total_gasto_passagens_padrao += (comprar_passagem_padrao * valor_da_passagem)
                quantidade_passagens_compradas_padrao += comprar_passagem_padrao
                saldo_restante_padrao -= valor_total_gasto_passagens_padrao

                print (f"\nParabéns! Você comprou {comprar_passagem_padrao} passagens.\n")
            else:
                print ("\nSaldo insuficiente!\n")
        else:
            print ("Por favor, digite um número maior do que 0.\n")

    except ValueError:
        print("Por favor, digite um número inteiro.")
```

Figura 6. Funcionalidade para comprar passagem.

Apresenta um submenu (menu_2) para selecionar o tipo de usuário. Você seleciona por exemplo, comprar passagens como usuário padrão, como no exemplo acima e solicita a quantidade de passagens, há a validação caso for um inteiro (int) e positivo. Calcula-se o custo total da compra (aplicando desconto de 50% para estudante/idoso e 80% para social). No caso, o usuário padrão paga o valor inteiro. Então, verifica-se se o saldo restante é suficiente. Se sim, atualiza-se o valor total gasto com passagens, a quantidade de passagens compradas e reduz o valor do saldo restante.

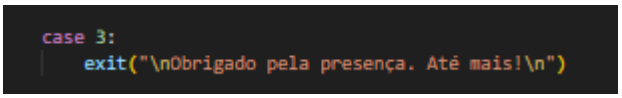


Figura 7. Funcionalidade para sair.

Utiliza a função exit() para encerrar o programa, exibindo uma mensagem de despedida.

2.3. Ordem de Codificação

Por um bom tempo, eu tentei realizar pequenos trechos do código, mas não deu muito certo. Ao longo do ciclo PBL, o momento em que mais consegui produzir foi quando estava ganhando um certo domínio das funcionalidades iniciais do Python. Tendo em vista isso, utilizei a seguinte ordem para codificar:

1. Inicialmente, tentei definir todas as variáveis do código de forma clara e objetiva porque precisava de um código funcional.
2. Depois, ao iniciar o programa, é necessário criar uma estrutura para definir um valor para passagem e esse foi o meu primeiro passo real, onde utilizei estruturas condicionais e variáveis, além de entradas e saídas. Fiz isso porque é o primeiro passo da elaboração desse programa.
3. Durante o fluxo, criei um menu principal com todas as funcionalidades para organizar melhor as informações e criar um sistema acessível. Comecei com o menu principal e fui destrinchando para submenus com cada categoria para permitir também maior facilidade ao utilizar o programa e não ser sobrecarregado de informações.

4. Na primeira funcionalidade (imprimir relatório), basicamente escrevi o comando de "print" para todas as variáveis necessárias no relatório e repeti esse código para em cada uma das funcionalidades ser possível acessar o relatório.
5. Na segunda funcionalidade, criei inicialmente uma estrutura condicional simples para o tratamento de dados e usei as variáveis para acumular outras variáveis, insight que descobri como executar somente depois de um período. Fiz o mesmo para cada categoria.
6. Na terceira funcionalidade, criei uma estrutura condicional simples para o tratamento de dados também e criei uma condição para verificar se o saldo restante é suficiente para comprar passagens. Utilizei as variáveis para acumular outras variáveis também e até mesmo tirar o acúmulo do saldo restante, já que você não pode comprar sem créditos.. Criei essa estrutura para cada variável novamente.
7. Por fim, utilizei a função "exit" para o caso do usuário querer encerrar o programa, já que ele funciona infinitamente.
8. Depois de tudo isso pronto, criei estruturas de repetição "while" e variáveis para evitar que o loop se repita infinitamente, tudo isso utilizando o sistema booleano. Coloquei uma delas na definição do valor da passagem para exibir somente uma vez esse trecho e a outra deixei no resto do código para funcionar infinitamente. A única maneira do código ter fim é com a funcionalidade "sair".
9. Como extra, utilizei estruturas de validação em todas as partes em que há uma entrada de dados para evitar erros indesejados caso o usuário digite algo incorreto.

O sistema piloto simples foi produzido utilizando a linguagem de programação Python 3.13.2, com o Visual Studio Code 1.98 como Ambiente Integrado de Desenvolvimento (*IDE*) e o sistema operacional Windows 10 Home Single Language.

3. Resultados e Discussões

Ao longo do programa, o que há de relevante é a etapa de definição do valor da passagem e logo após o redirecionamento para o menu principal. O programa se inicia com a definição do valor da passagem, onde um valor válido é inserido e depois disso aquele trecho não se repete até que o programa seja executado novamente. Depois disso, surge o menu principal, que possui 4 funcionalidades principais chamadas de "imprimir relatório", "fazer recarga", "comprar passagem" e "sair".

O usuário digita o número correspondente a respectiva função desejada e nas 3 possibilidades iniciais existe um submenu para o usuário selecionar a sua categoria e realizar o respectivo procedimento. No caso de “imprimir relatório”, informações como quantidade de recarga total, número de recargas, quantidade de passagens utilizadas, valor total gasto com passagens e saldo restante surgem de acordo com a categoria e esses dados vão se acumulando conforme o usuário usa naquela execução. Na parte de “fazer recarga”, o usuário simplesmente digita quanto ele quer recarregar em créditos e na de “comprar passagem” ele simplesmente digita quantas passagens para embarque ele quer comprar e se ele tiver saldo suficiente, a compra é efetuada. Algumas observações importantes são que o botão de sair encerra o programa, tanto o menu quanto o submenu funcionam com o usuário digitando o respectivo número da opção, toda vez que o usuário digitar algo incorreto ele será redirecionado para o menu principal, os descontos das passagens são calculados automaticamente e o programa não se encerra até que você saia.

Em relação ao conjunto de dados de entrada, na definição do valor da passagem ela é do tipo float (ponto flutuante), nos menus e submenus, ela é do tipo int (inteiro), ao fazer recargas ela é do tipo float (ponto flutuante) e ao comprar passagens ela é do tipo int (inteiro). Já em relação às saídas, temos as frases de que ações foram concluídas com sucesso ou não (como já explicado antes, o usuário retorna ao menu principal caso não tenha sucesso) e também temos o funcionamento das saídas de cada função. Ao fazer uma recarga, o valor tem que ser maior do que 0 e se isso for cumprido dados que estarão no relatório posteriormente serão acumulados (quantidade de recarga total acumula as recargas feitas, número de recargas acumula mais 1 toda vez que há uma recarga e saldo restante acumula a quantidade de recarga total), tudo isso para cada categoria. O mesmo acontece ao comprar uma passagem, pois o valor tem que ser maior do que 0 e também o saldo restante tem que ser maior ou igual a quantidade de passagens compradas vezes o valor respectivo da passagem para cada categoria. Se isso for cumprido, valor total gasto com passagens vai acumular o número de passagens compradas vezes o valor da passagem, o número de passagens compradas vai acumular a quantidade de passagens compradas ao longo do tempo e o saldo restante vai ser subtraído constantemente quando valores forem gastos com passagens. O relatório imprime todos esses dados citados, como já dito anteriormente.

3.1. Testes

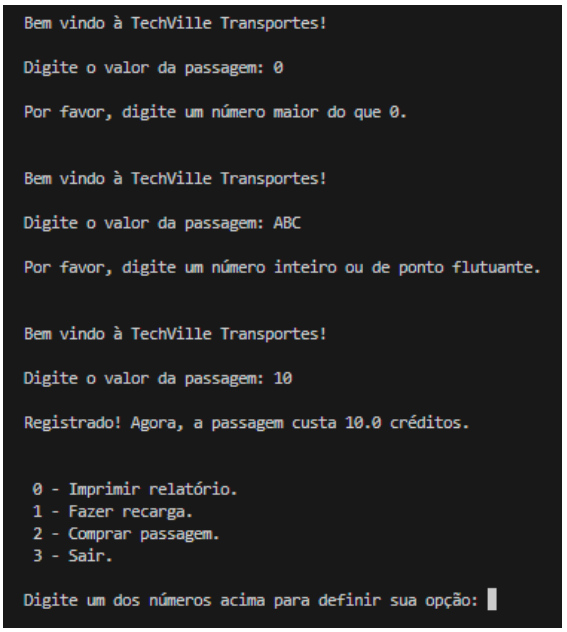


Figura 1. Teste de verificação da entrada na definição do valor da passagem.

Nesse teste, foi inserido um valor igual ou menor do que 0, um caractere inválido e um valor correto. Note que esse trecho do código não irá mais se repetir até uma nova execução e que uma mensagem de erro é exibida a cada caractere inadequado.

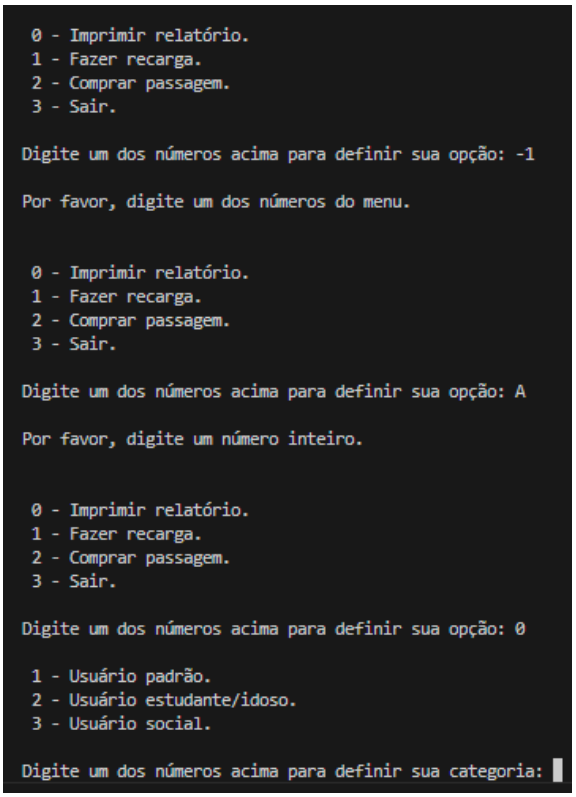


Figura 2. Teste de verificação da entrada no menu principal.

Nesse outro teste, também semelhante ao outro, mas agora feito no menu principal, mostra que ao inserir um caractere que não está nas opções uma mensagem de erro aparece. Note que novamente o menu principal se repete até algo válido ser inserido.

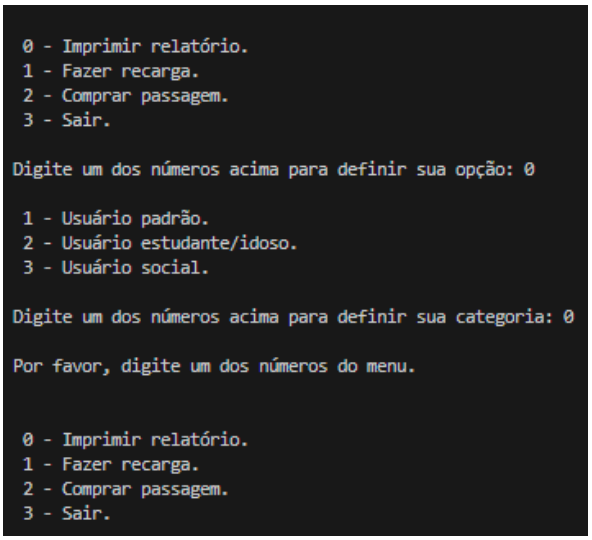


Figura 3. Teste de verificação da entrada em um submenu.

Agora, temos um dos submenus, onde ao inserir um caractere que não está nas opções, uma mensagem de erro é exibida, mas dessa vez ele retorna ao menu principal até o usuário digitar algo válido, ou seja, não se mantém no submenu.

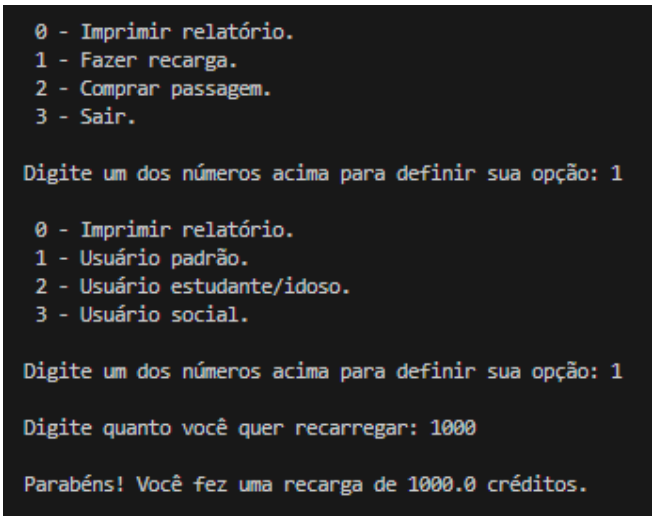


Figura 4. Teste de funcionalidade ao fazer uma recarga.

Em relação a questão de fazer uma recarga, contanto que o usuário insira algo válido (maior que zero e um número inteiro ou de ponto flutuante, já que o int vai ser convertido em float), o valor será recarregado e seu saldo será atualizado na respectiva categoria.

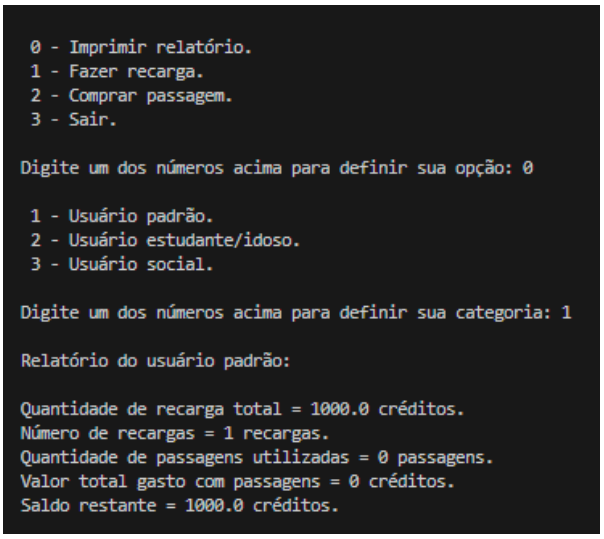


Figura 5. Teste de funcionalidade ao fazer imprimir o relatório.

Aqui, é mostrado somente o relatório sendo impresso.

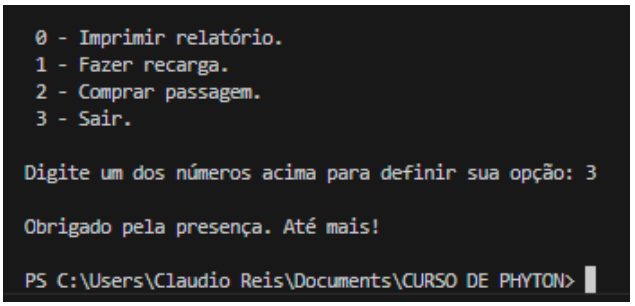


Figura 6. Teste de funcionalidade ao sair do programa.

Aqui, é mostrado o programa sendo encerrado.

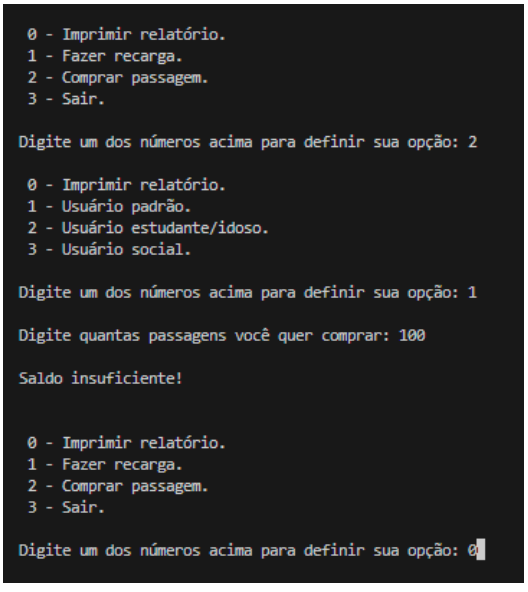


Figura 7. Teste de verificação de entrada ao comprar uma passagem.

Por fim, temos o teste de comprar uma passagem, onde se o usuário inserir uma quantidade de passagens maior do que zero e com um caractere válido, a quantidade de passagens desejadas são compradas se houver saldo suficiente. Se não houver, ele imprime uma mensagem de erro e retorna ao menu principal.

Com isso, nenhum erro aparente foi detectado e não existem situações conhecidas em que o programa não funcionaria.

4. Conclusão

Portanto, tendo em vista todo o conteúdo apresentado, acredito que a execução do sistema foi bem-sucedida e todos os requisitos foram cumpridos, talvez com algumas funcionalidades "extras", como a possibilidade de escolher exatamente o usuário que você deseja em qualquer uma das funções, a possibilidade do código funcionar infinitamente até o usuário decidir sair e a verificação que lida com possíveis erros de entrada.

Além disso, como possíveis melhorias para o programa, poderiam ser desenvolvidas as seguintes funcionalidades: interface do programa, cadastro de usuários específicos e únicos, criação de estruturas de repetição para o código não voltar para o menu principal sempre e a redução do código com estruturas de programação mais avançadas.

5. Referências Bibliográficas

DOWNEY, Allen B. Pense em Python: pense como um cientista da computação. 2. ed. São Paulo: Novatec, 2016. Disponível em: http://penseallen.github.io/PensePython2e/. Acesso em 17 de Março de 2025.

APÊNDICE A - Fluxograma Final do PBL

Fluxograma Final do PBL

Fonte: Autoria própria