Programação Funcional

Lucas Gomes Andrade: 2116273

Caio Moretti de Macedo: 2213905

Leonardo Alex Evaristo Chacon: 2213835

Gisele Da Costa Soares: 2213825

Lucas Aguiar Marques: 2124701

Fortaleza, 20 de março de 2024

GitHub: https://github.com/Caio-Moretti/AF-Prog-Funcional

Funções

Lucas Gomes: Código fonte

Caio Moretti: Testes

Lucas Aguiar : Requisitos

Leonardo Alex: Testes

Gisele da Costa: Documentação

Requisitos

Os requisitos foram feitos com a ajuda de chatbot (ChatGPT)

Funcionais:

Realizar operações matemáticas básicas por meios das funções adição, subtração, multiplicação e divisão dentro da função calculadora.

O sistema deve validar as entradas fornecidas pelo usuário por meio da verificação das entradas na função calculadora().

O sistema deve perguntar ao usuário se ele deseja executar outra operação, por meio da função outra_operação().

O sistema deve ser capaz de tratar erros que possam ocorrer durante as operações matemáticas, utilizando a classe Monad e seu método bind.

Não funcionais

O sistema deve ser fácil de usar com instruções claras e mensagens de erro compreensíveis.

O sistema deve ser projetado para que seja fácil a adição de outras operações. A estrutura do código permite adicionar facilmente futuras operações.

O código deve ser organizado para que seja fácil manter a manutenção do mesmo. Isso é possível por conta das funções específicas pra cada tarefa.

O sistema deve ser facilmente testável, demonstrado pela presença de testes unitários.

Caso de testes

Os testes foram corrigidos e ajustados com a ajuda de chatbots (ChatGPT e Gemini)

Os casos de teste para a calculadora foram feitos usando o PyTest e abrangem a validação de funcionalidades essenciais e o tratamento de entradas e erros:

- Testes de Operações Matemáticas: Verificam-se as funções de adição e operações gerais (adicionar, subtrair, multiplicar, dividir) retornam os resultados esperados para diferentes entradas.
- Teste do Menu: Confirma que o menu é criado corretamente, listando as quatro operações matemáticas básicas.
- Teste de Continuação de Operação: Avalia a função que pergunta ao usuário se deseja realizar outra operação, garantindo que responda corretamente as entradas 's' (sim) e 'n' (não).
- Testes de Monad: Testam a implementação da monad para manipulação de erros e resultados de operações, verificando tanto o comportamento em condições normais quanto em condições de erro.
- Testes de Cenários de Calculadora: Simulam o uso da calculadora através de entradas mockadas, testando a realização de somas, subtrações, multiplicações, divisões e a resposta a uma divisão por zero ou entrada inválida.
- Esses casos de teste asseguram a robustez da calculadora, garantindo que ela funcione corretamente em diferentes cenários e manipule adequadamente as entradas do usuário

Nossa cobertura de testes alcançou os 81% (pytest-cov)

```
----------- coverage: platform win32, python 3.9.0-final-0 -----------
Name      Stmts   Miss  Cover
---------------------------------
main.py      57     11    81%
---------------------------------
TOTAL        57     11    81%
```

Funções Implementadas

Monad

```python
    # Monad para tratar erros e resultados de cálculos
    class Monad:
        def __init__(self, valor, error=None):
            self.valor = valor
            self.error = error

        def bind(self, func):
            try:
                novo_valor = func(self.valor)
                return Monad(novo_valor)
            except Exception as e:
                return Monad(None, error="Erro")

        def get_value(self):
            if self.error:
                return self.error
            return self.valor

```

Continuação

```
70
71      # Função de continuação para imprimir o resultado
72      def imprimir_resultado(resultado):
73          print("O resultado é:", resultado)
74
```

Alta ordem

```
61
62      # Função de alta ordem para aplicar uma operação a dois números
63      def operacao(op, a, b):
64          return op(a, b)
65
66
```

Lambda

```
66
67      # Função lambda para adição
68      adicao = lambda x, y: x + y
69
70
```

Closure

```
47
48      # Closure para armazenar a última operação realizada
49      def operacao_anterior():
50          ultima_operacao = None
51
52          def salvar_operacao(op):
53              nonlocal ultima_operacao
54              ultima_operacao = op
55
56          def obter_operacao():
57              return ultima_operacao
58
59          return salvar_operacao, obter_operacao
60
```

Comprehension

```
36
37      # List comprehension para criar menu
38      def criar_menu():
39          return [f"{i}: {op}" for i, op in enumerate(["Soma", "Subtração", "Multiplicação", "Divisão"])]
40
41
```