Caixa Eletrônico com POO e Tkinter em Python

Seu Nome

9 de dezembro de 2024

Resumo

Este artigo apresenta o desenvolvimento de uma aplicação de caixa eletrônico utilizando a biblioteca *tkinter* em Python. O projeto integra conceitos de Programação Orientada a Objetos (POO) e manipulação de dicionários, fornecendo uma aplicação prática e educativa para reforçar esses conceitos. A aplicação permite aos usuários realizar operações bancárias básicas, como verificar saldo, depositar e sacar dinheiro.

Sumário

Introdução

Este artigo tem como objetivo desenvolver uma aplicação de caixa eletrônico utilizando a biblioteca *tkinter* em Python. A aplicação irá integrar conceitos de Programação Orientada a Objetos (POO) e manipulação de dicionários para fornecer uma experiência prática e educativa. Através deste projeto, os usuários poderão realizar operações bancárias básicas, como verificar saldo, depositar e sacar dinheiro.

1 Manipulação de Dicionários

1.1 Introdução aos Dicionários

Dicionários em Python são coleções de pares chave-valor que permitem acesso eficiente a dados baseados em chaves únicas. Eles são extremamente úteis para armazenar e manipular dados que possuem uma relação direta, como um nome e um telefone, ou um produto e seu preço.

1.1.1 Criação de Dicionários

Dicionários são criados utilizando chaves {} e pares chave-valor separados por dois pontos :. A chave pode ser de qualquer tipo imutável (como uma string, número ou tupla), enquanto os valores podem ser de qualquer tipo.

```
dados = {"nome": "Alice", "idade": 30, "cidade": "S o␣Paulo"}
```

1.1.2 Acesso a Valores

Para acessar os valores armazenados em um dicionário, utilizamos a chave correspondente entre colchetes [].

```
print(dados["nome"])  # "Alice"
```

1.1.3 Modificação de Valores

Podemos modificar os valores de um dicionário atribuindo um novo valor à chave correspondente.

```
dados["idade"] = 31
```

1.1.4 Remoção de Pares Chave-Valor

Para remover um par chave-valor de um dicionário, usamos a palavra-chave `del`.

```
del dados["cidade"]
```

1.2 Métodos Importantes dos Dicionários

1.2.1 Métodos Comuns

- `.keys()`: Retorna todas as chaves do dicionário.
- `.values()`: Retorna todos os valores do dicionário.
- `.items()`: Retorna todos os pares chave-valor do dicionário.
- `.get()`: Retorna o valor para a chave especificada, ou um valor padrão se a chave não for encontrada.
- `.pop()`: Remove um item do dicionário e retorna o seu valor.
- `.update()`: Atualiza o dicionário com os pares chave-valor de outro dicionário.

1.2.2 Exemplos

```
# Métodos básicos
print(dados.keys())      # dict_keys(['nome', 'idade'])
print(dados.values())    # dict_values(['Alice', 31])
print(dados.items())     # dict_items([('nome', 'Alice'), ('idade', 31)])

# Usando .get() para acessar valores com uma chave padrão
print(dados.get("cidade", "Desconhecido"))  # "Desconhecido"

# Usando .pop() para remover um item e obter seu valor
idade = dados.pop("idade")
```

```
print(idade)  # 31

# Usando .update() para atualizar o dicion rio
dados.update({"idade": 32, "telefone": "1234-5678"})
```

2 Programação Orientada a Objetos (POO)

2.1 Conceitos Básicos

Programação Orientada a Objetos (POO) é um paradigma de programação que utiliza "objetos"para representar dados e métodos. Os principais conceitos da POO incluem classes, objetos, atributos e métodos.

2.1.1 Classe

Classe é a estrutura fundamental da POO. Uma classe define um novo tipo de objeto contendo dados (atributos) e funções (métodos).

2.1.2 Objeto

Objeto é uma instância de uma classe. Cada objeto é uma realização específica da classe com valores reais para os atributos definidos pela classe.

2.1.3 Atributos

Atributos são variáveis que pertencem a uma classe e armazenam informações sobre os objetos criados a partir da classe.

2.1.4 Métodos

Métodos são funções que pertencem a uma classe e operam sobre seus atributos. Métodos são usados para definir os comportamentos dos objetos.

2.2 Exemplo de Classe e Objeto

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def saudacao(self):
```

```
        return f"Ol  , meu nome    {self.nome} e eu tenho {self.idade}

# Cria   o de um objeto
pessoa1 = Pessoa("Alice", 30)
print(pessoa1.saudacao())  # "Ol  , meu nome    Alice e eu tenho 30 an
```

2.3 Herança

Herança é um dos principais pilares da POO. Permite que uma nova classe herde atributos e métodos de uma classe existente. Isso promove o reuso de código e a criação de hierarquias de classes.

2.3.1 Exemplo de Herança

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def som(self):
        pass

class Cachorro(Animal):
    def som(self):
        return "Au Au!"

class Gato(Animal):
    def som(self):
        return "Miau!"

# Cria   o de objetos
cachorro = Cachorro("Rex")
gato = Gato("Mia")

print(cachorro.som())  # "Au Au!"
print(gato.som())      # "Miau!"
```

2.4 Encapsulamento

Encapsulamento é o princípio de esconder os detalhes internos dos objetos e expor apenas o que é necessário. Isso é feito definindo atributos como

privados e fornecendo métodos públicos para acessá-los e modificá-los.

2.4.1 Exemplo de Encapsulamento

```
class ContaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.__saldo = saldo  # Atributo privado

    def depositar(self, valor):
        self.__saldo += valor

    def sacar(self, valor):
        if valor <= self.__saldo:
            self.__saldo -= valor
        else:
            print("Saldo insuficiente")

    def mostrar_saldo(self):
        return self.__saldo

conta = ContaBancaria("Alice", 1000)
conta.depositar(500)
conta.sacar(200)
print(conta.mostrar_saldo())  # 1300
```

2.5 Polimorfismo

Polimorfismo permite que objetos de diferentes classes sejam tratados através da mesma interface. Em Python, isso é frequentemente implementado através de herança e métodos substituídos.

2.5.1 Exemplo de Polimorfismo

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def som(self):
        pass
```

```python
class Cachorro(Animal):
    def som(self):
        return "Au Au!"

class Gato(Animal):
    def som(self):
        return "Miau!"

# Fun  o que aceita qualquer objeto que implemente o m todo som
def fazer_som(animal):
    print(animal.som())

# Cria  o de objetos
cachorro = Cachorro("Rex")
gato = Gato("Mia")

fazer_som(cachorro)  # "Au Au!"
fazer_som(gato)      # "Miau!"
```

Desafio: Desenvolvendo um Caixa Eletrônico com POO e Tkinter em Python

Objetivo

Criar uma aplicação gráfica de caixa eletrônico que permita aos usuários realizar operações bancárias básicas como verificar saldo, depositar e sacar dinheiro. A aplicação deve utilizar *tkinter* para a interface gráfica, POO para a estruturação do código, e dicionários para armazenar os dados dos usuários.

Requisitos do Desafio

1. **Interface Gráfica:**
    - Use a biblioteca *tkinter* para criar a interface gráfica do caixa eletrônico.
    - A interface deve ter campos para entrada de dados do usuário (número da conta, valor) e botões para as operações (saldo, depósito, saque).

2. **Programação Orientada a Objetos:**
    - Defina classes para representar as contas bancárias.
    - Implemente métodos para realizar operações como verificar saldo, depositar e sacar.
3. **Manipulação de Dicionários:**
    - Use dicionários para armazenar os dados das contas (por exemplo, número da conta como chave e saldo como valor).

Estrutura do Projeto

```
caixa_eletronico/
                main.py
                README.md
```

Descrição dos Componentes

1. **Classe `ContaBancaria`:**
    - Define uma conta bancária com métodos para depositar, sacar e mostrar o saldo.
    - Usa encapsulamento para proteger o saldo da conta.
2. **Dicionário `contas`:**
    - Armazena instâncias de `ContaBancaria` usando o número da conta como chave.
3. **Funções de Interface:**
    - `verificar_saldo()`, `depositar()` e `sacar()`: Manipulam as operações bancárias utilizando os métodos da classe `ContaBancaria`.
4. **Interface Gráfica com *tkinter*:**
    - Cria uma janela com campos de entrada para o número da conta e o valor.
    - Botões para verificar saldo, depositar e sacar dinheiro.

Desafio Adicional

- Adicione a funcionalidade de criação de novas contas através da interface gráfica.
- Implemente validações adicionais, como verificar se o valor inserido é positivo

3 Referências

- Python Software Foundation. (2023). Python Documentation. Disponível em: <https://docs.python.org/3/>
- Tkinter Documentation. (2023). Disponível em: <https://docs.python.org/3/library/tkinter.html>
- Programação Orientada a Objetos. (2023). Disponível em: <https://pt.wikipedia.org/wiki/ProgramaÃğÃčo_orientada_a_objetos>