Sistema de Gerenciamento de Biblioteca

Uma Abordagem Simples em Python

Implementação de um sistema completo para gerenciamento de acervo e empréstimos

Estrutura do Código: Classes

Livro

codigo: str titulo: str autor: str
ano_publicacao: int genero: str
quantidade_total: int
quantidade_disponivel: int

Usuario

id_usuario: str
nome: str tipo: str

Emprestimo

id_usuario: str
codigo_livro: str
dia_emprestimo: int
dia_devolucao_prevista: int
status: str
dia_devolucao_efetiva: int = 0
multa: float = 0.0

Funções Auxiliares Essenciais

⌨️ entrada(prompt, tipo=str, erro="Entrada inválida", validacao=None)

```
def entrada(prompt, tipo=str, erro="Entrada inválida", validacao=None): while True: try: v =
tipo(input(prompt).strip()) if not v or (validacao and not validacao(v)): print(erro) else: return v except:
print(erro)
```

Função robusta para coletar e validar entrada do usuário:

- Converte a entrada para o tipo especificado
- Aplica uma função de validação personalizada
- Exibe mensagem de erro quando necessário
- Persiste até receber uma entrada válida

buscar(lista, chave, campo)

```
def buscar(lista, chave, campo): return next((x for x in lista if getattr(x, campo) == chave), None)
```

Função genérica para buscar objetos em listas:

- Utiliza getattr() para acessar atributos dinamicamente
- Implementa o padrão generator expression para eficiência
- Retorna o primeiro objeto correspondente ou None
- Usada em todo o sistema para localizar livros, usuários e empréstimos

Funcionalidades: Livros

cadastrar_livro()

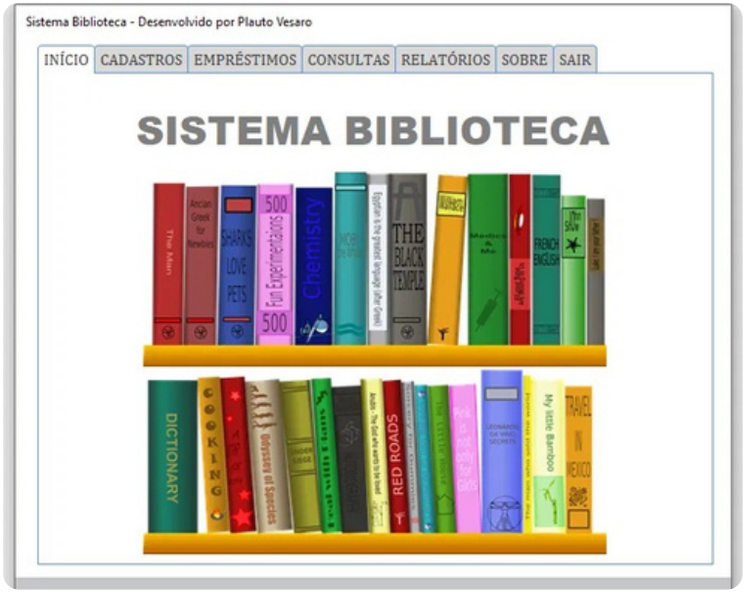
Adiciona um novo livro ao acervo, verificando a unicidade do código e coletando todos os dados necessários. Utiliza a função **entrada()** para validar cada campo.

listar_livros()

Exibe todos os livros cadastrados com seus detalhes completos, incluindo código, título, autor, ano, gênero e quantidades (total e disponível).

buscar_livro()

Permite a busca de livros por código, título ou autor, utilizando uma busca case-insensitive com **termo in campo.lower()**. Exibe os resultados encontrados com informações resumidas.



Funcionalidades: Usuários

cadastrar_usuario()

Registra um novo usuário no sistema, verificando:

- Unicidade do ID de usuário
- Validação do tipo de usuário (aluno/professor)
- Coleta de dados pessoais

```
usuarios.append( Usuario( id_u, entrada("Nome: "),
entrada("Tipo (aluno/professor): ", str, "Tipo inválido",
lambda x: x in ["aluno", "professor"]) ) )
```

listar_usuarios()

Apresenta a lista de todos os usuários cadastrados com seus dados:

- ID único do usuário
- Nome completo
- Tipo de usuário (aluno/professor)

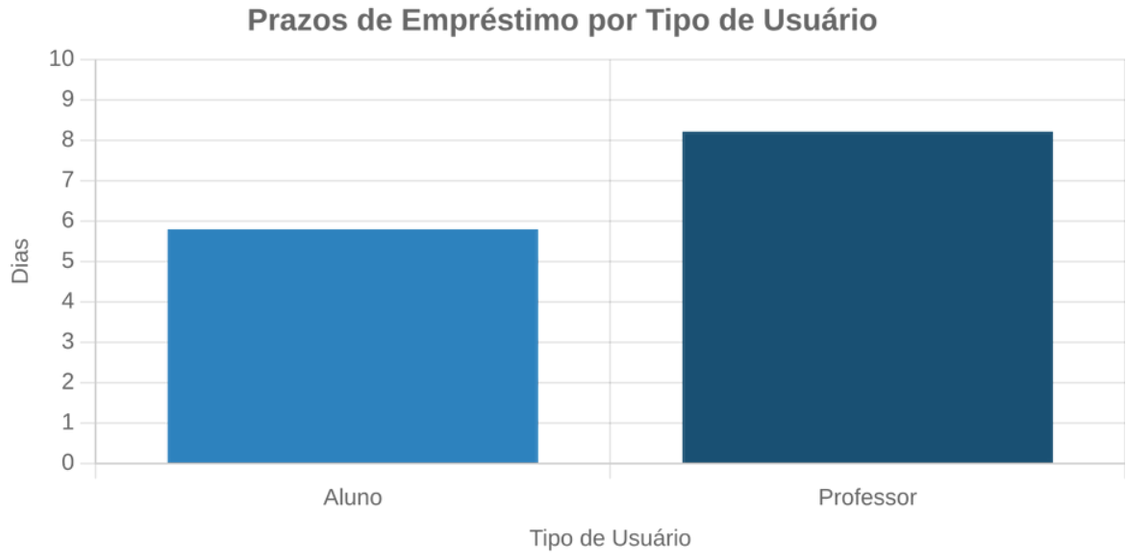
```
for u in usuarios: print(f"{u.id_usuario} | {u.nome} |
{u.tipo}")
```

Diferenças por Tipo de Usuário

O tipo de usuário (aluno ou professor) determina o prazo de empréstimo:

- Alunos: 7 dias de prazo
- Professores: 10 dias de prazo

```
# Na função emprestar(): prazo = 7 if u.tipo == "aluno" else
10
```



Funcionalidades: Empréstimos e Devoluções

emprestar()

Realizao empréstimode um livro a um usuário:

- Solicita ID do usuário e código do livro
- Verifica existência do usuário e do livro
- Verifica disponibilidade do livro
- Define prazo com base no tipo de usuário
- Atualiza quantidade disponível do livro
- Registra o empréstimo na lista

```
prazo = 7 if u.tipo == "aluno" else 10
l.quantidade_disponivel -= 1 emprestimos.append(
Emprestimo(id_u, cod, dia_atual_sistema, dia_atual_sistema +
prazo, "ativo") )
```

devolver()

Processa a devolução de um livro:

- Solicita ID do usuário e código do livro
- Localiza o empréstimo ativo correspondente
- Registra a data de devolução efetiva
- Atualiza o status do empréstimo para "devolvido"
- Incrementa a quantidade disponível do livro
- Calcula multa por atraso, se houver

```
emp.dia_devolucao_efetiva = dia_atual_sistema emp.status =
"devolvido" l.quantidade_disponivel += 1 atraso = max(0,
emp.dia_devolucao_efetiva - emp.dia_devolucao_prevista)
emp.multa = atraso * VALOR_MULTA_POR_DIA
```

Relatórios e Gerenciamento de Tempo

Função relatorios()

Gera relatórios sobre o estado atual da biblioteca:

✓Lista todos os empréstimos ativos

✓Identifica empréstimos atrasados

✓Calcula dias de atraso para cada empréstimo

✓Exibe informações de livros e usuários relacionados

```
ativos = [e for e in emprestimos if e.status == "ativo"] #
... atrasados = [e for e in ativos if
e.dia_devolucao_prevista < dia_atual_sistema] # ...
dias_atraso = dia_atual_sistema - e.dia_devolucao_prevista
```

Gerenciamento de Tempo

A função menu_gerenciar_tempo() permite simular a passagem do tempo no sistema:

✓Avançar 1 dia

✓Avançar 7 dias (1 semana)

✓Avançar N dias personalizados

✓Consultar o dia atual do sistema

A variável global dia_atual_sistema é fundamental para:

✓Registrar a data de empréstimo

✓Calcular a data prevista de devolução

✓Verificar atrasos e calcular multas

Implementação

```
def menu_gerenciar_tempo(dia_sistema_atual_param): # ... if opcao_tempo
== '1': dia_sistema_atual_param += 1 elif opcao_tempo == '2':
dia_sistema_atual_param += 7 elif
opcao_tempo == '3': n_dias = int(input("Quantos dias? "))
dia_sistema_atual_param += n_dias # ... return
dia_sistema_atual_param
```

Fluxo Principal e Menu

Função main()

O programa é executado através de um menu interativo na função main():

```
def main(): global dia_atual_sistema while True: print("\n=== Menu Principal ===") print(" 1.Livros\n 2.Usuários\n 3.Empréstimo\n 4.Devolução\n 5.Relatórios\n 6.Tempo\n 7.Sair") m = input("Opção: ") match m: case '1': # Submenu de Livros case '2': # Submenu de Usuários case '3': emprestar() case '4': devolver() case '5': relatorios() case '6': dia_atual_sistema = menu_gerenciar_tempo(dia_atual_sistema) case '7': print("Encerrando..."); break case _: print("Opção inválida.")
```

Características do fluxo principal:

- Utiliza match case (Python 3.10+) para controle de fluxo
- Implementa submenus para Livros e Usuários
- Acesso direto às funções principais
- Gerenciamento de variáveis globais

Estrutura de Menus

1. Livros (Submenu)

2. Usuários (Submenu)

3. Empréstimo

4. Devolução

5. Relatórios

6. Tempo

7. Sair

Considerações Finais

Pontos Fortes do Sistema

Simplicidade e Clareza

- ✓ Código direto e fácil de entender, ideal para fins didáticos
- ✓ Uso de dataclasses para representação concisa de entidades
- ✓ Funções bem definidas com responsabilidades claras
- ✓ Interface de linha de comando intuitiva

Extensibilidade

- ✓ Estrutura modular com classes e funções separadas
- ✓ Funções auxiliares reutilizáveis (entrada, buscar)
- ✓ Fácil adição de novas funcionalidades
- ✓ Possibilidade de expansão para interface gráfica

Melhorias Potenciais

- ✓ Persistência de dados (salvar/carregar em arquivo ou banco de dados)
- ✓ Interface gráfica para melhor experiência do usuário Validações mais
- ✓ robustas e tratamento de exceções Implementação de autenticação e
- ✓ níveis de acesso

Julho/2025

Obrigado, pela atenção!

Nome: Henrique Silveira Vicente